A MARSHALL CAVENDISH **36** COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE

UK £1.00 Republic of Ireland £1.25 Malta 85c Australia $2.25 New Zealand $2.95

INPUT

Vol. 3 **No 36**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front cover, Dave King. Pages 1113, 1114, 1115, 1116, 1117, Grant Symon. Pages 1119, 1120, 1122, 1124, Michael Strand. Pages 1126, 1142, 1143, Berry Fallon Design. Pages 1127, 1128, 1130, Alistair Graham. Pages 1132, 1134, Dave King. Page 1137, Advertising Arts. Pages 1136, 1137, 1141, Peter Reilly. Pages 1138, 1140, 1142, 1144, Kate Charlesworth.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Typeset by MS Filmsetting Limited, Frome, Somerset. Printed by Cooper Clegg Web Offset Ltd, Gloucester and Howard Hunt Litho, London.

There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

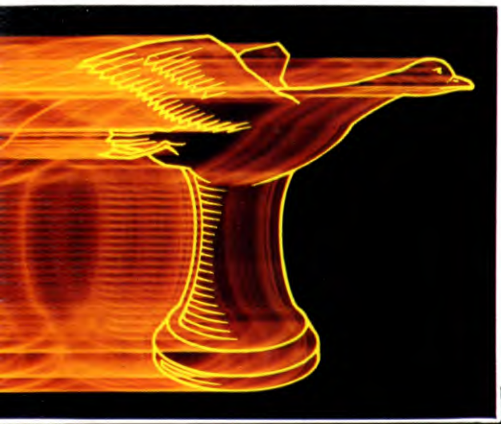
SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

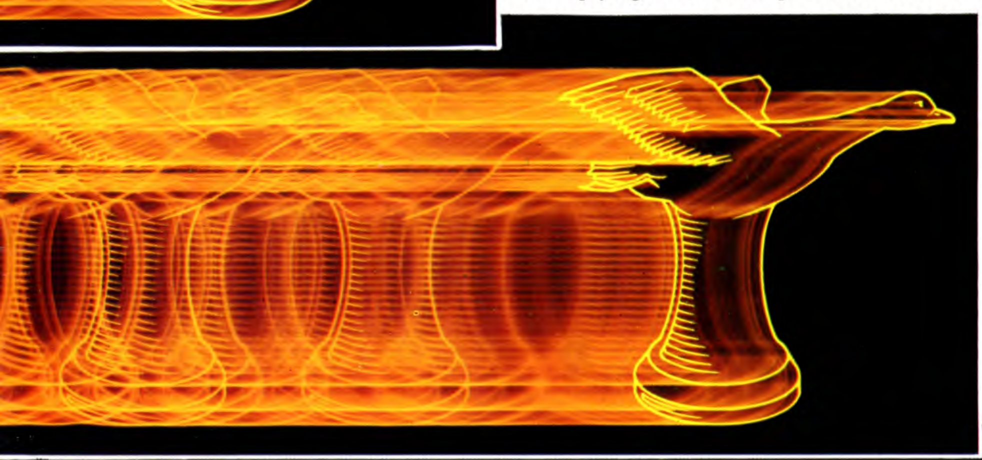
FOX AND GEESE GAME-2

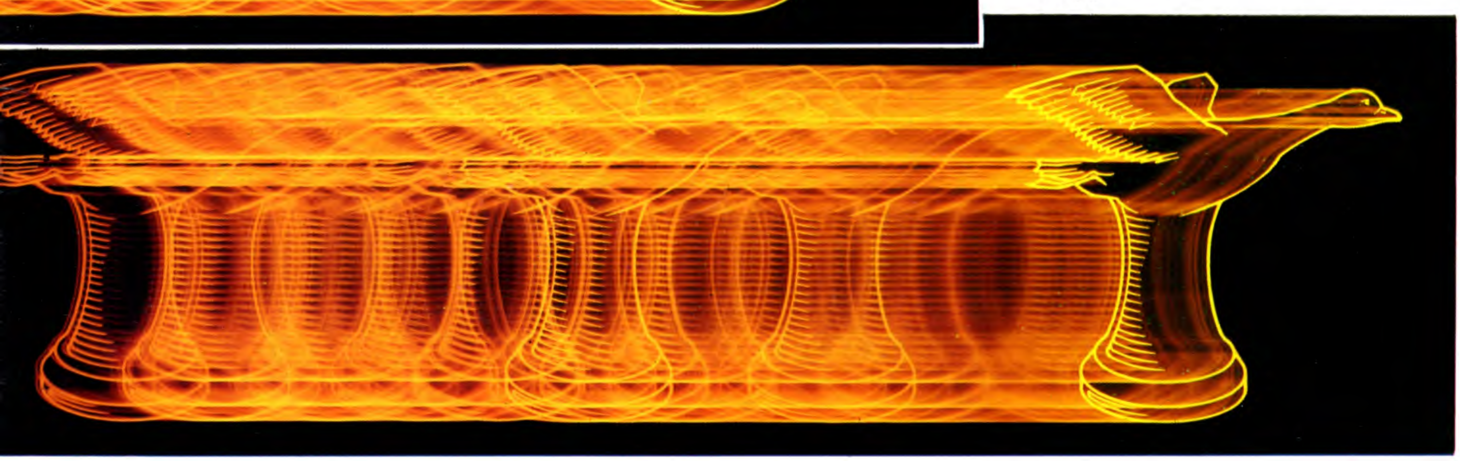
Use the last part's theory to start writing the Fox and Geese game. Here are the routines to initialize the game, and to map moves

This time enter initialization routines, and the vital mapping routine. You can also offer the player another go, but at this stage RUNning the program will be no use, as there are still many important routines to add.

In the next part of Fox and Geese you'll enter the thinking routines.







OVERVIEW

The program works by evaluating each position in the game according to the configuration of the pieces. Each position is given a numeric value by the program, so when looking ahead, the program is able to choose the best move by looking for the outcome with the highest value.

The program works in three ways when looking ahead. In its crudest workings (level one) it only looks one move ahead—it is a so-called 'one-mover'. At the higher levels of play it uses the alpha-beta algorithm to save time in searching through the ever-multiplying branches of possibilities. At intermediate levels the program looks through all the possibilities open to it.

The routines from Line 2010 to Line 3000 are only executed once, so they have been placed at the end of the program. With these seldom-used routines placed here, the main routines can be placed near the front of the program for speed—see pages 921 to 927.

INITIALIZATION

Here are the routines for all machines which are used to initialize the game. Arrays are DIMensioned, and FuNctions are DEFined. Three machines define the board graphics:

S

```
2010 DIM G(4): DEF FN U(A)=INT
   (A-4*INT (A/4)): DEF FN V(A)=INT
   (A-8*INT (A/8))>=4: DEF FN
   W(A)=INT (A-2*INT (A/2))
2015 LET HF=0: LET HG=0
2020 DIM B(32): LET B(1)=1: FOR I=1 TO
   31: LET B(I+1)=B(I)*2: NEXT I
2026 LET BX=B(32)*2-B(25): LET
   E=1E30: LET H=-1E30
2030 LET L2=LN (2)
2040 DIM B$(16,2,16): DIM G$(2,4): LET G$
   (1)="■■□□":LET G$(2)="■■"
   +CHR$ 146+CHR$ 147: DIM H$(2,4):
   LET H$(1)="□□■■": LET H$
   (2)=CHR$ 146+CHR$ 147+"■■"
2050 LET X=1: FOR A=1 TO 2: FOR B=1
   TO 2: FOR C=1 TO 2: FOR D=1 TO 2:
   LET B$(X,1)=G$(D)+G$(C)+G$(B)+
   G$(A)
```

```
2060 LET B$(X,2)=H$(A)+H$(B)+H$
    (C)+H$(D): LET X=X+1: NEXT D:
    NEXT C: NEXT B: NEXT A
2070 DIM S$(8,16): GOSUB 6000
2090 DIM F$(2,4): LET F$(1)="■■"+
    CHR$ 144+CHR$ 145: LET F$(2)=
    CHR$ 144+CHR$ 145+"■■"
2095 DEF FN C(B)=FN U(B-1)*(4-8*FN
    V(B-1))+12*FN V(B-1)
```

```
3 PRINT "▣▣▣▣";CHR$(8)
4 X=16:FOR Z=0 TO 15:POKE 646,Z:IF
    Z=6 THEN NEXTZ
5 X=X-1:POKE646,Z:PRINTTAB(5+X)
    "◪FOX AND GEESE◪":NEXTZ
10 POKE 53272,19:GOTO2010
2010 DIMG(4)
2020 DIMB(31):B(0)=1:FORI=1TO31:
    B(I)=B(I-1)*2:NEXTI
2026 BX=B(31)*2-B(24):E=1E30:H=
    -1E30
2030 L2=LOG(2):DEFFNA(F)=INT(LOG(F)/
    L2+.001)
2040 DIMB$(15,1):DIMG$(1):G$(0)="□
    □□□":G$(1)="□□◪☒"
2045 DIMH$(1):H$(0)="□□□□":H$
    (1)="◪☒□□"
2050 X=0:FORA=0TO1:FORB=0TO1:
    FORC=0TO1:FORD=0TO1:B$(X,0)=
    G$(D)+G$(C)+G$(B)+G$(A)
2060 B$(X,1)=H$(A)+H$(B)+H$(C)+
    H$(D):X=X+1:NEXTD,C,B,A
2070 DIMS$(7):FORA=0TO6STEP2:FORB=
    A*4TOA*4+3
2080 S$(A)=S$(A)+"□□"+RIGHT$
    (STR$(B),2)
2081 S$(A+1)=RIGHT$(STR$(B+4),2)+
    "□□"+S$(A+1)
2082 NEXTB,A
2090 DIMF$(1):F$(0)="□□◙♣":
    F$(1)="◙♣□□"
2095 DEFFNC(B)=(3ANDB)*
    (4-2*(4ANDB))+3*(4ANDB)
```

```
30 GOTO 2010
2010 DIMG(4)
2020 DIMB(31):B(0)=1:FOR I=1 TO 31:
    B(I)=B(I-1)*2:NEXTI
2026 BX=B(31)*2-B(24):E=1E30:H=-
    1E30
2027 L2=LOG(2)
2030 DEF FNA(F)=INT(LOG(F)/L2+.001)
2040 DIMB$(15,1):DIMG$(1):G$(0)=
    CHR$228+CHR$229+
    CHR$32+CHR$32:G$(1)=CHR$
    228+CHR$229+CHR$224+CHR$225
2045 DIMH$(1):H$(0)=CHR$32+
    CHR$32+CHR$228+CHR$229:H$(1)=
    CHR$224+CHR$225+CHR$228+
    CHR$229
2050 X=0:FOR A=0 TO 1:FOR B=0 TO
    1:FOR C=0 TO 1:FORD=0 TO
    1:B$(X,0)=G$(D)+G$(C)+G$(B)+
    G$(A)
2060 B$(X,1)=H$(A)+H$(B)+H$(C)+
    H$(D):X=X+1:NEXTD,C,B,A
2070 DIMS$(7):FOR A=0 TO 6 STEP2:
    FORB=A*4TOA*4+3
2080 S$(A)=S$(A)+CHR$230+CHR$231
2081 IFB<10 THEN S$(A)=S$(A)+
    "□"+STR$(B) ELSE S$(A)=S$(A)+
    STR$(B)
2083 S$(A+1)=CHR$230+CHR$231+S$
    (A+1)
2084 IF (B+4)<10 THENS$(A+1)=
    "□"+STR$(B+4)+S$(A+1)
    ELSE S$(A+1)=STR$(B+4)+
    S$(A+1)
2087 NEXTB,A
2090 DIMF$(1):F$(0)=CHR228+CHR$
    229+CHR$226+CHR$227:F$(1)=
    CHR$226+CHR$227+CHR$228+
    CHR$229
2095 DEFFNC(B)=(3 AND B)*(4-2*(4 AND
    B))+3*(4 AND B)
```

```
10 GOTO 2010
2010 DIM G(4),B(31),M(3,31),X(31),Z(31)
2020 B(0)=1:FORK=1TO31:B(K)=
    B(K-1)*2:NEXT
2026 BX=B(31)*2-B(24):E=1E30:H=1
    E30
2030 L2=LOG(2):DEFFNA(F)=INT(LOG
    (F)/L2+.001)
```

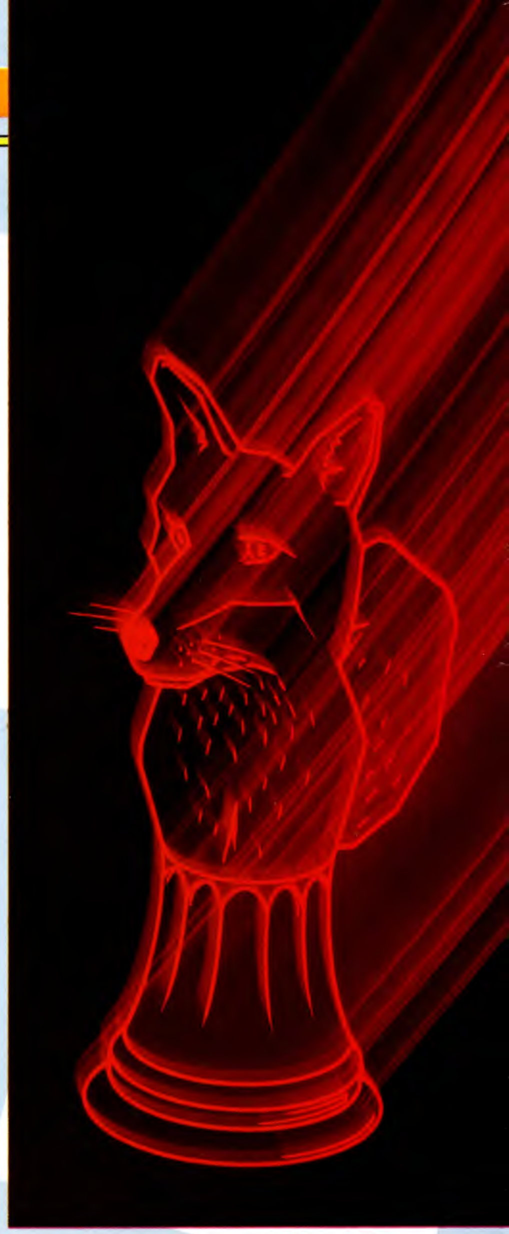
Line 2010 DIMensions the array used for storing the positions of the geese. Line 2020 numbers each square on the board that will be used in the game.

Line 2030 sets up the number of configurations that can be evaluated by the program. 0.001 has been added when DEFining FuNction A to prevent rounding errors when taking LOGarithms. Array B$, DIMensioned in Line 2040, is used for displaying rows on the board complete with pieces. F$ is used for the fox piece and blank square, and H$ for the geese and blank square. S$ is used to set up the numbers on the squares.

The Dragon and Tandy program does not have this section of program because the graphics board is set up in high resolution graphics.

AT THE START

This routine allows the player to choose who plays what, and to select the computer's skill level. Not too difficult at first!



```
2700 LET F=2: LET G(1)=29: LET
    G(2)=30: LET G(3)=31: LET G(4)=32:
    GOSUB 2710: GOTO 1010
2710 CLS : PRINT AT 0,9; INK 1;"FOX AND
    GEESE": INPUT "DO YOU WANT ...";TAB
    5;"TO PLAY FOX ? (y/n)";I$
2720 LET PF=0: IF I$="Y" OR I$="y"
    THEN GOTO 2760
2730 LET PF=1: IF I$<>"N" AND
    I$<>"n" THEN GOTO 2710
2740 INPUT "LEVEL OF FOX SKILL? ";SF: IF
    SF<1 OR SF>10 THEN GOTO 2740
2750 LET HF=131*(SF=5)+613*(SF=6)+
    1997*(SF>6)
2760 INPUT "DO YOU WANT□";TAB 5;"TO
    PLAY GEESE? (y/n)";I$
2770 LET PG=0: IF I$="Y" OR I$="y"
    THEN GOTO 2860
2780 LET PG=1: IF I$<>"N" AND
    I$<>"n" THEN GOTO 2760
2790 INPUT "LEVEL OF GEESE SKILL□"; SG:
    IF SG<1 OR SG>10 THEN GOTO 2790
```

```
2800 LET HG=131*(SG=5)+613*
   (SG=6)+1997*(SG>6): IF HF<HG
   THEN LET HF=HG
2860 INPUT "DO YOU WANT TO ALTER THE
   □□□□□□□□STARTING POSITION
   ? ";I$: IF I$="N" OR I$="n" THEN
   GOTO 3000
2880 IF I$<>"Y" AND I$<>"y" THEN
   GOTO 2860
2890 GOSUB 210: GOSUB 310: INPUT "DO
   YOU WANT TO MOVE FOX ? ";I$
2900 IF I$="N" OR I$=n" THEN GOTO
   2930
2910 IF I$<>"Y" AND I$<>"y" THEN
   GOTO 2890
2920 INPUT "MOVE FOX TO□";F: IF F<1
   OR F>32 THEN GOTO 2920
2930 FORG=1 TO 4: GOSUB 210: GOSUB
   310
2940 INPUT "DO YOU WANT TO MOVE
   GOOSE AT ";(G(G));" ? ";I$
2950 IF I$="N" OR I$="n" THEN GOTO
   2990
2960 IF I$<>"Y" AND I$<>"y" THEN
   GOTO 2940
2970 INPUT "MOVE GOOSE TO ";I: IF FN X(I)
   OR I=F THEN GOTO 2960
2972 IF I<1 OR I>32 THEN GOTO 2970
2980 LET G(G)=I
2990 NEXT G: IF FN X(F) THEN PRINT
   "THERE IS A GOOSE UNDER THE FOX":
   FOR I=1 TO 1500: NEXT I:
   GOTO 2910
3000 RETURN
```

```
2500 DIM R(1999),S(1999)
2700 F=1:G(1)=28:G(2)=29:G(3)=30:
   G(4)=31:GOSUB2710:GOTO1010
2710 PRINT"♥DO YOU WANT TO PLAY FOX
   ((Y/N)?"
2720 GET I$:PF=0:IFI$="Y"THEN2760
2730 PF=1:IFI$<>"N"THEN2720
2740 SF=0:INPUT"♥LEVEL OF FOX SKILL
   (1-10)";SF:IFSF<1ORSF>10THEN2740
2750 HF=-131*(SF=5)-613*(SF=6)-
   1997*(SF>6)
2760 PRINT"♥DO YOU WANT TO PLAY
   GEESE (Y/N)?"
2770 GET I$:PG=0:IFI$="Y"THEN2860
2780 PG=1:IFI$<>"N"THEN 2770
2790 SG=0:INPUT"♥LEVEL OF GEESE
   SKILL (1-10)";SG:IFSG<1ORSG>10
   THEN2790
2800 HG=-131*(SG=5)-613* (SG=6)
   -1997*(SG>6):IFHF<HGTHEN HF=HG
2860 PRINT"♥DO YOU WANT TO ALTER
   THE STARTING"
2870 PRINT "POSITION (Y/N)?"
2875 GET I$:IF I$="N"THEN3000
2880 IFI$<>"Y"THEN2875
2890 GOSUB210:GOSUB310:PRINT"DO YOU
   WANT TO MOVE THE FOX (Y/N?□";
2900 GET I$:IFI$="N"THEN2930
2903 IF I$<>"Y"THEN2900
2915 PRINT "Y"
2920 INPUT"■■■■■■■■
   MOVE FOX TO";F:IFF<0ORF>31
   THEN 2920
2925 GOSUB340
2930 FORG=1TO4:GOSUB210:GOSUB310
2940 PRINT "DO YOU WANT TO MOVE THE
   GOOSE AT";G(G):PRINT"(Y/N)?□";
2950 GET I$:IFI$="N"THEN2950
2960 IFI$<>"Y"THEN2990
2965 PRINT "Y"
2970 INPUT"■■■■■■■■MOVE
   GOOSE TO";I:GOSUB340
2971 IFFNX(I)ORI=FTHENPRINTTAB(8);
   "ALREADY OCCUPIED":GOTO2940
2972 IFI<0ORI>31THEN2970
2980 G(G)=I
2990 NEXTG:IFFNX(F)THEN PRINTTAB(8);
   "THERE IS A GOOSE UNDER THE FOX"
2995 FORI=1TO1500:NEXTI
3000 RETURN
```

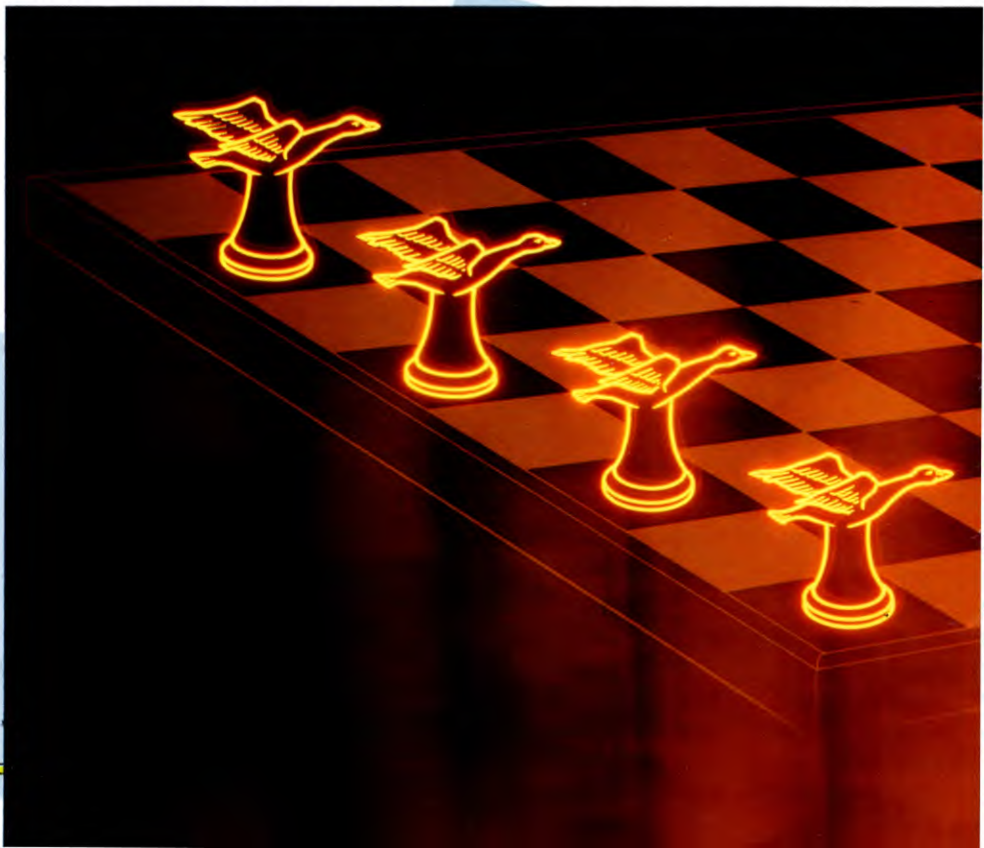
```
2700 F=1:G(1)=28:G(2)=29:G(3)=30:
   G(4)=31:GOSUB2710:GOTO1010
2710 CLS:PRINT"DO YOU WANT TO PLAY
   FOX (Y,N) ?"
2720 I$=GET$:PF=0:IFI$="Y" THEN
   2760
2730 PF=1:IFI$<>"N" THEN2710
2740 SF=0CLS:INPUT"LEVEL OF FOX SKILL
   (1-10)" SF:IF SF<1 OR SF>10 THEN
   2740
2750 HF=-131*(SF=5)-613*(SF=6)-
   947*(SF>6)
2760 CLS:PRINT"DO YOU WANT TO PLAY
   THE GEESE (Y/N) ?"
2770 I$=GET$:PG=0:IF I$="Y" THEN
   2860
2780 PG=1:IF I$<>"N" THEN2760
2790 CLS:INPUT"LEVEL OF GEESE SKILL
   (1-10)"SG:IFSG<1ORSG>10THEN2790
2800 HG=-131*(SG=5)-613*(SG=6)-
   947*(SG>6):IF HF<HG THEN HF=HG
2860 CLS:PRINT"DO YOU WANT TO ALTER
   THE STARTING"
2870 PRINT"POSITION (Y/N)?"
2875 I$=GET$:IF I$="N" THEN 3000
2880 IFI$<>"Y" THEN2860
2890 GOSUB210:GOSUB310:PRINT"DO YOU
   WANT TO MOVE THE FOX (Y,N)? ";
2900 I$=GET$: IF I$="N" THEN 2930
2910 IF I$<>"Y" THEN 2890
2915 PRINT"Y"
2920 INPUTTAB(8)"MOVE FOX TO" F:IF
   F<0 OR F>31 THEN 2920
2925 GOSUB340
2930 FOR G=1 TO 4:GOSUB210:GOSUB310
2940 PRINT"DO YOU WANT TO MOVE THE
   GOOSE AT□";G(G):PRINT"(Y/N)?";
2950 I$=GET$:IF I$="N" THEN 2990
2960 IFI$<>"Y" THEN 2940
2965 PRINT"Y"
2970 INPUTTAB(8)"MOVE GOOSE TO□"I
2971 IF FNX(I) OR I=F THEN PRINT
   TAB(8)"ALREADY OCCUPIED":GOTO
   2940
2972 IF I<0 OR I>31 THEN 2970
2980 G(G)=I
2990 NEXTG:IFFNX(F) THEN PRINT
   TAB(8)"THERE IS A GOOSE UNDER THE
   FOX":FOR I=1 TO 1500:NEXTI:GOTO
   2890
3000 RETURN
```

```
2500 DIM R(1500),S(1500)
2700 F=1:G(1)=28:G(2)=29:G(3)=30:
   G(4)=31:GOSUB2710:GOTO1010
2710 CLS:PRINT "DO YOU WANT HUMAN TO
   PLAY FOX□□□(Y/N)□?";
2720 K$=INKEY$:IF K$<>"Y" AND
   K$<>"N" THEN 2720
2730 PRINTK$:PF=1:IF K$="Y" THEN
   PF=0:GOTO2760
2740 PRINT:PRINT "LEVEL OF FOX SKILL
   (0-9)□?";
2745 K$=INKEY$:IFK$<"0"ORK$>"9"
   THEN2745
2746 SF=VAL(K$)+1:PRINTK$
2750 HF=-131*(SF=5)-613*(SF=6)-
   1499*(SF>6)
2760 PRINT:PRINT"□DO YOU WANT
   HUMAN TO PLAY GEESE (Y/N)□?";
2770 K$=INKEY$:IF K$<>"Y" AND
   K$<>"N" THEN 2770
2780 PRINTK$:PG=1:IF K$="Y" THEN
   PG=0:GOTO2860
2790 PRINT:PRINT"LEVEL OF GEESE SKILL
   (0-9) ?";
2795 K$=INKEY$:IF K$<"0" OR K$>"9"
   THEN2795
2796 SG=VAL(K$)+1:PRINTK$
2800 HG=-131*(SG=5)-613*(SG=6)-
   1499*(SG>6):IF HF<HG THEN HF=HG
2860 PRINT:PRINT" □DO YOU WANT TO
   CHANGE THE□□□□□□□□
   STARTING POSITIONS (Y/N)?";
2870 K$=INKEY$:IF K$<>"Y" AND
   K$<>"N" THEN 2870
2880 IF K$="N" THEN3000
2890 GOSUB210
2920 DRAW"BM180,80"+MW$:XX=
   FNXX(1):YY=FNYY(1):GOSUB1810:F=4*
   INT(YY/20):F=FNCN(F)
2925 PUT(68,8)-(87,27),SQ,PSET:PUT
   (XX,YY+5)-(XX+19,YY+13),FX,PSET
2930 FORG=1TO4:GOSUB210
2940 XX=FNXX(G(G)):X1=XX:YY=FNYY
   (G(G)):Y1=YY:GOSUB1810:PUT
   (X1,Y1)-(X1+19,Y1+19),SQ,PSET
2950 I=4*INT(YY/20):I=FNCN(I)
2960 IF(FNX(I) ORI=F)ANDI<>G(G)
   GOSUB5000:GOTO2940
2970 PUT(XX,YY+5)-(XX+19,YY+14),GS,
   PSET:G(G)=I
2990 NEXT:IF FNX(F) GOSUB5000:GOTO2920
2995 C=1:G=G(1)
3000 RETURN
```

Line 2700 sets the starting position, with the four geese occupying the four squares at the bottom of the board, and the fox occupying the second square from the left on the top row.

After Line 2700 has initialized the starting position of the fox and the geese, Lines 2710 to 2750 give the player the option of playing fox, and prompt for a skill level from one to ten if the computer is going to play fox. Lines 2760 to 2800 are similar, except the player is given the option of playing geese.

The game has been designed to allow adjustment of the starting position, either allowing you to continue where you left off last time (you will need to take note of the positions of the pieces when the game ended, or to try winning (or losing!) from a particularly interesting position. The lines from 2860 to 3000 ask if the player wants to alter the starting position, give prompts, and make sure that the positions chosen are legal.



MAPPING MOVES

The mapping moves routine is one of the most important in the program.

```
14Ø DEF FN X(B)=B=G(1) OR B=G(2) OR
    B=G(3) OR B=G(4)
21ØØ DIM R$(8,16)
2142 DEF FN Z(B)=(B=G(1))+
    (B=G(2))*2+(B=G(3))*3+(B=G
    (4))*4
215Ø DIM M(4,32): DIM X(32): DIM Z(32)
216Ø FOR B=1 TO 32: LET
    U=B-1-4*INT (B/4-.2): FOR A=1
    TO 4: LET M(A,B)=(B-2)-2*U+8*
    ((B<5) OR(A>2))+(A*7-6)*(U=3)+
    (A=2)+(A=4): NEXT A: LET X(B)=
    ((B>4)+(B<29))*((U<3)+1): LET
    Z(B)=(B>4)*((U<3)+1): NEXT B
218Ø DIM V(11):DIM A(11):DIM F(11):
    DIM P(11):DIM C(11):DIM R(1):DIM S(1)
```

```
21ØØ DIMR$(7)
211Ø DEFFNF(B)=((B>3)+(B<28))*
    (((3ANDB)<3)-1)-1
212Ø DEFFNG(B)=(B>3)*(((3ANDB)<3)-
    1)-1
213Ø DEFFNM(A)=B-2*(3ANDB)-2-8*
    (B<4ORA>1)-(1+A*7)*
    ((3ANDB)=3)+(1ANDA)
214Ø DEFFNX(B)=(B=G(1)OR B=G(2)OR
    B=G(3)OR B=G(4))
2142 DEFFNZ(B)=-(B=G(1))-(B=G
    (2))*2-(B=G(3))*3-(B=G(4))*4
215Ø DIM M(3,31),X(31),Z(31)
216Ø FOR B=Ø TO 31:FOR A=Ø TO 3: M(A,
    B)=FNM(A): NEXTA: X(B)=FNF (B):
    Z(B)=FNG (B): NEXT B
```

```
21ØØ DIMR$(7)
211Ø DEFFNF(B)=((B>3)+(B<28))*(((3
    AND B)<3)-1)-1
212Ø DEFFNG(B)=(B>3*)(((3ANDB)
    <3)-1)-1
213Ø DEFFNM(A)=B-2*(3 AND B)-2-8*
    (B<4 OR A>1)-(1+A*7)*((3 AND
    B)=3)+(1 AND A)
214Ø DEFFNX(B)=(B=G(1) OR B=G(2)
    OR B=G(3) OR B=G(4))
2142 DEFFNZ(B)=-(B=G(1))-(B=G
    (2))*2-(B=G(3))*3-(B=G(4))*4
215Ø DIMM(3,31),X(31),Z(31)
216Ø FORB=ØTO31:FORA=ØTO3:M(A,B)=
    FNM(A):NEXTA:X(B)=FNF(B):Z(B)=
    FNG(B):NEXTB
218Ø DIMP(1Ø),V(1Ø),F(1Ø),A(1Ø),C(1Ø)
25ØØ DIMR(95Ø),S(95Ø):HF=Ø
```

```
211Ø DEFFNF(B)=((B>3)+(B<28))*
    (((3ANDB)<3)-1)-1
212Ø DEFFNG(B)=(B>3)*(((3ANDB)<3)-
    1)-1
214Ø DEFFNX(B)=(B=G(1) OR (B=G(2)
    OR (B=G(3) OR B=G(4))
2142 DEFFNZ(B)=-(B=G(1))-(B=G
    (2))*2-(B=G(3))*3-(B=G(4))*4
215Ø DEFFNXX(B)=-((7ANDB)<4*)
    (28+4Ø*(3ANDB))-((7ANDB)>3)*
    (128-4Ø*(3ANDB))
2155 DEFFNYY(B)=8+2Ø*INT(B/4)
2156 DEFFNCN(B)=B-((7ANDB)<4)*
    (XX-28)/4Ø-((7ANDB)>3)*(128-XX)
    /4Ø
216Ø FORB=ØTO31:FORA=ØTO3:M
    (A,B)=B-2*(3ANDB)-2-8*
    (B<4ORA>1)-(1+A*7)*
    ((3ANDB)=3)+(1ANDA):NEXT:X(B)=
    FNF(B):Z(B)=FNG(B):NEXT
```

Lines 211Ø to 216Ø build the map of fox and geese moves in array M. Alongside this map, the number of possible fox moves, array X, and the number of possible goose moves, array Z, are also set up. The arrays are copies of the functions in Lines 211Ø to 2142. the Spectrum routine is shorter because of the way the machine's logic works.

ANOTHER GO?

Now add an 'another go?' routine.

```
141Ø INPUT "ANOTHER GAME (Y,N) ? ";I$
142Ø IF I$="Y" OR I$="y" THEN GOTO
    27ØØ
143Ø IF I$<>"N" AND I$<>"n" THEN
    GOTO 141Ø
144Ø STOP
```

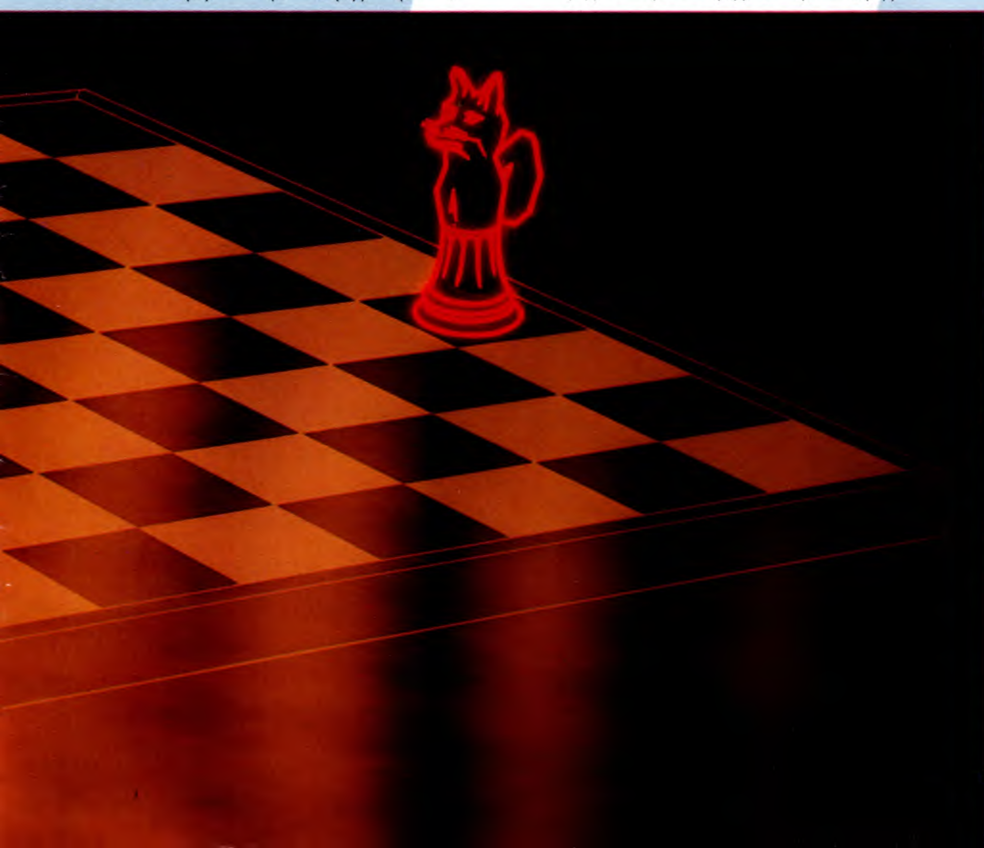
```
141Ø PRINTTAB(8);"ANOTHER GAME (Y/N)?"
142Ø GET I$:IFI$="Y"THEN27ØØ
143Ø IFI$<>"N"THEN142Ø
144Ø PRINT "": POKE 53272,21:END
```

```
141Ø PRINTTAB(8);"ANOTHER GAME (Y/N)?"
142Ø I$=GET$:IF I$="Y" THEN 27ØØ
143Ø IF I$<>"N" THEN 141Ø
144Ø CLS:END
```

```
141Ø PRINT@39Ø, "ANOTHER GAME (Y/N)?"
142Ø K$=INKEY$:IF K$="Y" GOSUB4Ø4Ø:
    CLS:GOTO27ØØ
143Ø IF K$<>"N" THEN 142Ø
144Ø CLS:END
```

These lines should be familiar by now, and they come into play when the geese manage to trap the fox, or the fox manages to reach the opposite end of the board.

Do not try to RUN the program at this stage, as there are many vital parts of the program still to add. In the next part of the article you'll add the routines which will allow you to play the game.



STARTING WITH SPREADSHEETS

If you find yourself having to deal with lots of figures, then it's a good time to enlist your micro's help—it's probably a lot better with numbers than you are

One numerical chore that afflicts most people is keeping track of their own expenditure, and *INPUT*'s accounts program on pages 136 to 145 provides one way for the micro user to sort out where the money is going. But now, we look at a different system that is modelled on the one used by professional accountants—the spreadsheet.

Spreadsheets are among the most versatile of all programs, with almost unlimited potential for handling numerical information. And they are by no means restricted just to financial data.

This article is in three parts. To start with, there is a look at what a spreadsheet can do, and what they are used for. Then, you will be able to program your own simple spreadsheet, using the listing which starts this time. You will get detailed instructions on putting it to work for you, in a later part.

WHAT IS A SPREADSHEET?

Spreadsheets utilise one of the biggest advantages of a computer—its ability to make calculations very quickly. In essence, even the biggest of computers is simply a complex adding machine. In fact, a computer can only deal with numbers, as those who have dipped into machine code will endorse.

The computerized spreadsheet can be an immensely powerful tool. It is normally used for financial accounting but it can be used to build all sorts of computer models. It replaces the old pencil, paper and calculator methods, used by accountants for forecasting a company's profits or research scientists investigating population growth. And at domestic level it can be used to keep track of personal expenditure, or details relating to a hobby.

An accountant's traditional spreadsheet, used for recording revenue and expenditure, for instance, consists of a large sheet of paper, usually taking up a double page spread. It is divided horizontally into rows and vertically into columns. This produces a grid of boxes or 'cells'. Along the top the accountant usually enters the months of the year so that each column refers to one month. Down the side of the grid are headings such as revenue and expenditure. For more detailed analysis he may introduce sub-headings such as home sales, exports, labour costs, raw material costs, overheads and so-on. Each row then refers to a specific area of revenue or expenditure.

The final heading down the side of the grid is usually Profit/Loss and the figures at the end of each column show how much profit or loss has been made each month. At the end of each row, in the thirteenth column, the total revenue or expenditure for each specific area over the whole year is recorded.

Filling in the cells with figures is a laborious task, whether or not a computer is used. But accountants who use a paper spreadsheet also face the laborious task of calculating the Profit/Loss figures. This means adding up all the revenue figures, all the expenditure figures and then subtracting total expenditure from total revenue.

ENTER THE COMPUTER

In many respects, the computer spreadsheet is just like the one used in the paper system, with the same grid, divided into columns and rows. In practice, to produce cells of a reasonable size, only a small section of the whole spreadsheet is displayed on the screen, which can be used to 'window' the particular area in which you are interested.

Once again, as in the paper version, you can enter what you like into the blank cells, which have no special meaning until you define them. You can type in a label or heading, or enter figures, depending upon what you want the spreadsheet to display.

So far, the computerized spreadsheet is, if anything, a little more cumbersome than a sheet of paper. But its real power is the ability to manipulate the information that you have fed into it. Hidden under the blank spreadsheet on which you make entries is another spreadsheet. This one tells the computer what to do with the information that it finds in each cell. In fact, the 'hidden' spreadsheet is no mystery, because you have also put this there, and it is available to view or modify at any time.

To go back to our struggling accountant, let's say that he wants one column to display an item's cost, the next to show a percentage of tax payable on that sum, and the third column to add the first two together. Using the computer, he can program the computer to do this on demand. All that's necessary is to set up an instruction in each of the cells in column two, telling the computer to multiply the number in column one by a fixed percentage. A similar instruction in each of the cells in column three will then get the computer to calculate the required total, by adding the contents of the relevant cells in the previous columns.

FORECASTING THE FUTURE

Another problem for the accountant with his large sheet of paper, is coping with changes. An increase in labour costs, for instance, would mean recalculating the total expenditure figure and subtracting it again from the total revenue figure to find the revised Profit/Loss figure. If you are simply recording figures the task is not so onerous but if you are making forecasts for a year or more ahead it could mean hundreds of recalculations. This is the sort of job that's time consuming, boring and prone to errors if carried out manually, even with the aid of a calculator. A computer can accomplish that sort of task in a few milliseconds.

As long as you have entered the figures correctly onto the spreadsheet—not always the simple task it may sound!—a change to one figure will automatically produce the appropriate adjustments to all other related figures.

If the figure in the raw material cost row is changed, for instance, the total cost will be adjusted accordingly and the necessary changes made to the total revenue. This is the simplest of all examples and some spreadsheets are capable of carrying out enormously complex calculations. This makes them very useful for answering the 'What if...' questions which constantly need to be answered in business—and in many other areas. Although used mainly for business purposes, a spreadsheet can also be used to predict, for example, population changes. In fact, any situation where there are many interdependent variable values is a suitable application for a spreadsheet.

The power and versatility of spreadsheets has led to them becoming the biggest selling type of software. Many spreadsheets are compatible with other software so that it is possible to build up a complete library of software with more serious uses. A word processor, a database management system and a spreadsheet would make up an enormously versatile 'suite' of programs.

A TYPICAL SPREADSHEET

The basic unit of the spreadsheet is the cell. The contents of each cell can either be a string variable—a word, for instance—a number or a formula. When the spreadsheet is loaded into the computer the cells are a certain preset width. On some spreadsheets this 'default' can be changed, either at the outset or at a later stage.

The value displayed in each cell can either be a number that's been entered or the result of a calculation. The number of rows and columns will vary from spreadsheet to spreadsheet but there are commonly 65 columns and 256 rows in the serious business spreadsheets. That's 16640 individual cells—a lot for any micro to handle! *INPUT*'s spreadsheet has 24 columns and between 20 and 30 rows depending on the computer.

The cells are always addressed and located by letters or numbers along the x and y axes of the grid on the sheet but exactly how varies from spreadsheet to spreadsheet. Most use a combination of letters and numbers with columns labelled A, B, C ... Z, and then AA, AB, AC ... AZ and so on for large spreadsheets. The rows in such a case would be numbered from 1 onwards. This is the method used in the programs below.

Various commands are available to enter equations, values or labels, to copy cells, or to look at different parts of the sheet. Other commands perform the calculations and allow you to load and save the data. The equations can cope with all the usual mathematical operations—plus, minus, multiply and divide—as well as percentages and the total in any row or column. The cursor is normally used to move around the spreadsheet. The cursor highlights a whole cell at a time and this cell becomes the 'active' cell. It is the one which you are now working on and which will be directly affected by your instructions to the computer. This is how the Spectrum works. The other computers use a different method where each cell is specified first and its contents entered at the bottom of the screen before being transferred to the correct position on the sheet.

PLANNING AND DESIGN

The first stage of using a spreadsheet is one of the most difficult, often requires a great deal of planning and doesn't involve the use of the computer! Before you start you must decide exactly what you want the computer to do, because this will affect how you design your spreadsheet. A properly planned spreadsheet is an ideal method of displaying information clearly and concisely. But, as is so often the case in computing, your spreadsheet will only be as good as you make it. A sloppy approach to the task will lead to an untidy, muddled spreadsheet, difficult to read, hiding information rather than revealing it.

As a practical example, let's say you want to design a spreadsheet to help with domestic finance over the year. This will obviously use the months of the year as the title of each column along the top of the sheet. But deciding what the title of each row will be is more difficult.

First of all , how detailed do you want it to be? Mortgage, Rates, Fuel, House/Contents, Insurance and Maintenance are obvious titles referring to your house. But do you want to treat expenditure on home improvements as an independent category? Or do you want to include the running costs of the car—Petrol, Tax, Insurance, Service, Repairs—in a joint category and call it something like General Expenditure. It really depends on how much detailed information you want.

A spreadsheet can be particularly useful for keeping track of the value of your assets, such as car and house. You ought to be able to find out by what sort of percentage your house is appreciating and your car is depreciating in value.

Working out the annual increase in the value of your house looks quite simple initially. One year after you have bought it the value will be the price you paid for it mulitplied by the annual percentage increase in value—P*X%—where X is the percentage increase plus 1ØØ. For example, X would be 1ØØ·5% for annual increase of ·5%. The formula to work out the value in the second year is P*X%*X%. In the third year the formula gets even longer and by the end of ten year's it's impossible to handle.

With a spreadsheet there is an easier way. Thankfully you do not need to be a mathematician, familiar with dozens of mathematical formulae, to be able to use the spreadsheet to its full potential. In a case like this you can usually use the address of one cell to refer to the contents of that cell. In this instance the formula gets no more complicated than P*X% where P is the contents of the previous cell. In the formula you would be writing for the spreadsheet, P would actually be the address of the previous cell and might look something like B1Ø*1ØØ·5%.

If the formula is entered in cell C1Ø the answer is displayed in that cell. Entering the formula C1Ø*1ØØ·5% in D1Ø tells the computer to take the number displayed in C1Ø and multiply it by 1ØØ·5%. The actual form the equations have to take varies from spreadsheet to spreadsheet and the programs below use a rather different method. However, the exact details will all be explained in the instructions on how to use the program that will be coming later.

Using the address of each cell instead of the contents of that cell makes working with a spreadsheet very easy. It enables almost anyone to carry out very complicated mathematical tasks with the aid of a little bit of common sense and patience. Care must be taken when referring from one cell to another, however. You must not, for example, use cell B1Ø in a formula in C1Ø while the formula in B1Ø depends on the result obtained in C1Ø! The computer cannot work out the result of either one until it has solved the other!

If the spreadsheet failed to take account of this then the program would crash as the computer attempted to resolve the paradox.

WHAT IF. . . ?

Your domestic budget spreadsheet will enable you to answer all sorts of 'What if...' questions. What if the mortgage rate rises by 2% in June? What if we buy a bigger car?

In fact, the last example points to another area where spreadsheets can be used other than for financial forecasting and budgeting. The difference between central heating systems using different fuels can be illustrated at a glance. As long as you can estimate how much heat loss you would prevent by using double glazing you could work out the how much you would save and how long it would take to recover the cost of installation.

Although even the simplest spreadsheet can be used for quite complicated serious applications, spreadsheets can also be fun. Models other than the usual financial models can be built. At the simplest level, and just for fun, it is possible to create a circular reference through cells which will carry on forever.

There are enormous variations from spreadsheet to spreadsheet. As a general rule the more powerful the spreadsheet, the more expensive it will be, and the bigger the micro needed to run it. A small spreadsheet might have half a dozen commands and a similar number of functions. Compare that with Multiplan's 20 main commands and 40 functions. With sophisticated spreadsheets it is possible to introduce statements with a function similar to some Basic commands such as IF . . . THEN, AND, OR and NOT. In other words it is possible to program the spreadsheet.

ENTERING THE PROGRAM

The spreadsheet program is quite long, so it is given in three parts. Enter the lines given below now, and save them so the remaining lines can be added later. Instructions on how to use the program will also be given with the following two parts.

Each A, B and C should be entered in graphics mode.

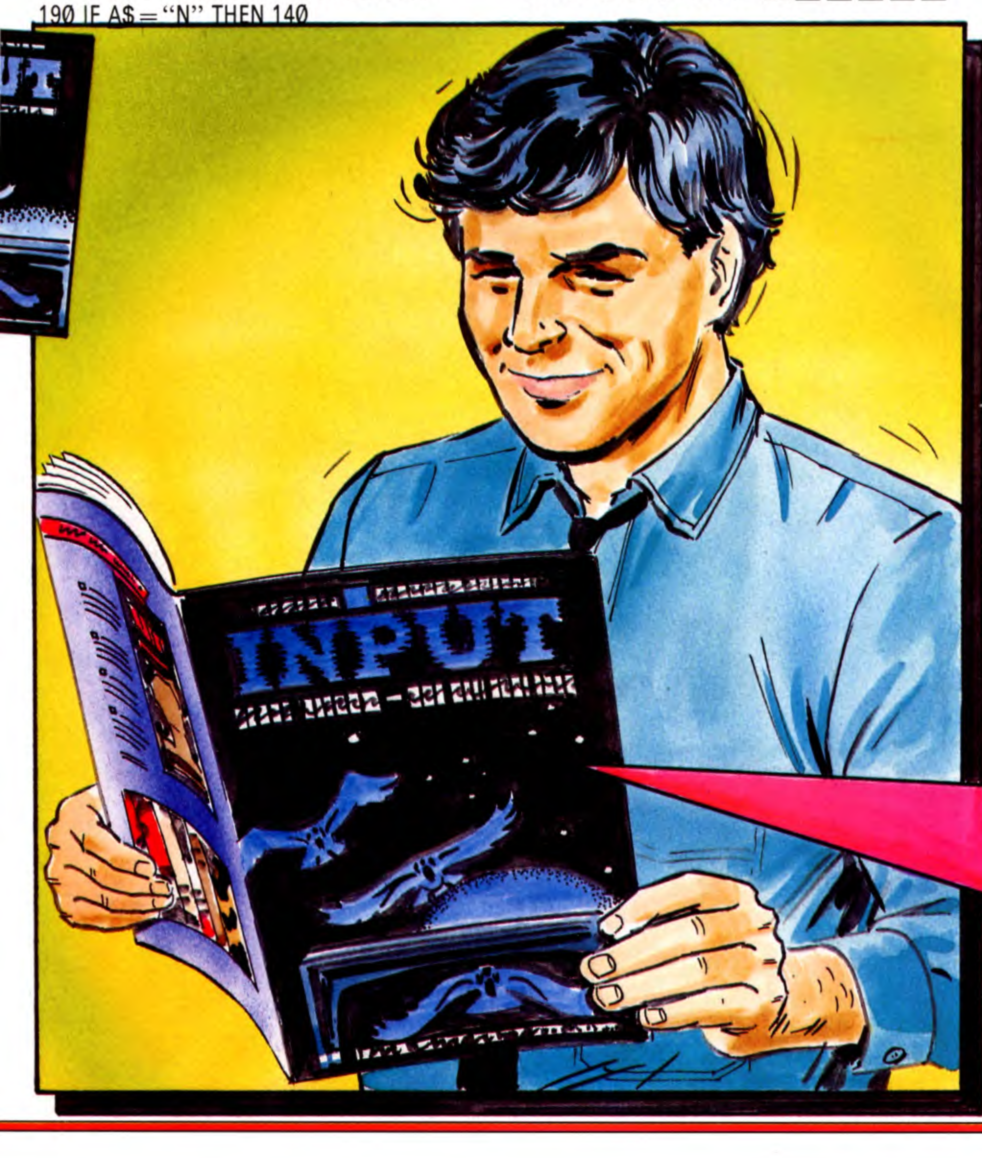
S

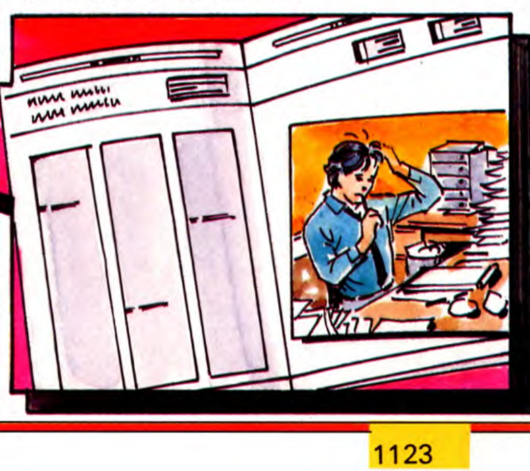
```
5 BORDER Ø: PAPER Ø: INK 7: CLS
10 DIM b$(11): DIM s$(8): DIM
   d$(3Ø,24,18): DIM v(4): DIM z$(5,4)
2Ø GOSUB 173Ø: POKE 23658,8: LET
   t$="VAL": LET os=Ø: LET sflag=Ø: LET
   wx=1: LET wy=1: LET cx=1: LET cy=1
3Ø CLS: PRINT "□□□□A□□□□□
   □□□A□□□□□□□□A□□□
   □□□□□A": FOR x=4 TO 32 STEP
   9: FOR y=2 TO 21 STEP 2:PRINT AT y,x;
   "A": NEXT y: NEXT x: FOR y=1 TO 21
   STEP 2: PRINT AT y,Ø;"BBBBCBBBBBBBB
   CBBBBBBBBCBBBBBBBBC": NEXT y
4Ø FOR X=Ø TO 2: PRINT AT
   Ø,9*x+9;CHR$ (wx+x+64): NEXT x
5Ø FOR x=Ø TO 9: PRINT AT
   (x+1)*2,1;("□" AND
   wy+x<1Ø);wy+x: NEXT x
6Ø PRINT AT Ø,Ø;t$;"□": FOR y=Ø TO 9:
   FOR x=Ø TO 2: GOSUB 123Ø: NEXT x:
   NEXT y: PRINT #1;AT Ø,Ø;"□□□□□
   □□□□□□□□□□□□□□□
   □□□□□□□□□□□"
7Ø PRINT AT cy*2,((cx-1)*9)+5; BRIGHT 1;
   FLASH 8; PAPER 8; INK 8; OVER 1;"□"
8Ø IF INKEY$="'" AND wy>1 THEN LET
   wy=wy-1Ø: GOTO 4Ø
9Ø IF INKEY$="&" AND wy<2Ø THEN LET
   wy=wy+1Ø: GOTO 4Ø
1ØØ IF INKEY$="(" AND wx<21 THEN LET
   wx=wx+3: GOTO 4Ø
11Ø IF INKEY$="%" AND wx>1 THEN LET
   wx=wx-3: GOTO 4Ø
12Ø PRINT AT cy*2,((cx-1)*9)+5; FLASH 8;
   BRIGHT Ø; INK 8; PAPER 8; OVER 1;"□"
13Ø LET cy=cy+(INKEY$="6" AND
   cy<1Ø)-(INKEY$="7" AND cy>1):
   LET cx=cx+(INKEY$="8" AND
   cx<3)-(INKEY$="5" AND cx>1)
14Ø IF INKEY$="i" OR INKEY$="I" THEN
   GOSUB 125Ø
15Ø IF INKEY$="v" OR INKEY$="V" THEN
   LET t$="VAL": GOTO 6Ø
16Ø IF INKEY$="e" OR INKEY$="E" THEN
   LET t$="EQU": GOTO 6Ø
17Ø IF INKEY$="?" THEN PRINT AT Ø,Ø;
```

```
    FLASH 1;"CALC": GOSUB 810: GOTO 60
180 IF INKEY$="z" OR INKEY$="Z" THEN
    PRINT AT 0,0; FLASH 1;"COPY": GOSUB
    230: GOTO 60
190 IF INKEY$="P" OR INKEY$="p" THEN
    COPY
200 IF INKEY$="NOT□" THEN GOSUB
    1780: GOTO 30
210 IF INKEY$="—" THEN GOSUB 1840:
    GOTO 30
220 GOTO 70
230 PRINT #1;AT 0,0;"CELL TO COPY ? ":
    LET d=1: LET c=3: LET x=15: GOSUB
    580: GOSUB 670: IF f THEN BEEP .2,30:
    GOTO 230
240 PRINT #1;AT 0,0;"ABS OR REL (A OR
    R)? ": LET x=22: LET d=2: LET c=1:
    GOSUB 580: GOSUB 670: IF f THEN BEEP
    .2,30: GOTO 240
250 PRINT #1;AT 0,0;"COL OR ROW (C OR
    R)? ": LET x=22: LET d=3: LET c=1:
    GOSUB 580: GOSUB 670: IF f THEN BEEP
    .2,30: GOTO 250
260 PRINT #1;AT 0,0;"FROM CELL NO ?":
    LET x=16: LET d=4: LET c=3: GOSUB
    580: GOSUB 670: IF f THEN BEEP .2,30:
    GOTO 260
270 PRINT #1;AT 0,0;"TO CELL NO ? ": LET
    x=14: LET d=5: LET c=3: GOSUB 580:
    GOSUB 670: IF f THEN BEEP .2,30: GOTO
    270
280 GOSUB 770: IF NOT f THEN GOTO 320
290 PRINT #1;AT 0,0;"COMMAND ERROR
    :PRESS A TO ABORT OR ANY OTHER KEY
    TO RE-ENTER"
300 PAUSE 10: PAUSE 0: PRINT #1;AT
    0,0;"□□□□□□□□□□□□□□□□
    □□□□□□□□□□□□□□□□
    □□□□□□□□□□□□□□□□
    □□□□□□□□□□□□□□□□
    □□□□□": IF INKEY$="a"OR
    INKEY$="A"THEN RETURN
310 GOTO 230
320 LET a=(CODE z$(1,2))-64: LET
    b=VAL z$(1,3 TO (VAL z$(1,1)+1)): LET
    s$=(d$(b,a,9 TO 16) AND
    t$="EQU")+(d$(b,a, TO 8) AND
    t$="VAL"): LET c$=d$(b,a,17): LET
    z=CODE d$(b,a,18)
330 IF z$(2,2)="R" AND T$="EQU" AND
    C$="1" THEN GOTO 390
340 FOR a=fc TO tc: FOR b=fr TO tr
350 IF t$="EQU" THEN LET d$(b,a,9 TO
    16)=s$: LET d$(b,a,17)=c$: LET
    d$(b,a,18)=CHR$z
360 IF t$="VAL" THEN LET d$(b,a, TO
    8)=s$
370 NEXT b: NEXT a
380 RETURN
390 LET s$=d$(b,a,9 TO 16): GOSUB 890:
    LET a=(CODE z$(4,2)-64)-(CODE
    z$(1,2)-64): LET b=VAL z$(4,3 TO (VAL
    z$(4,1)+1))-VAL z$(1,3 TO (VAL
    z$(1,1)+1))
400 LET v(2)=v(2)+b-1: LET
    v(4)=v(4)+((b-1) AND v(3)<>26)
410 LET v(3)=v(3)+((a-1) AND
    v(3)<>26): LET v(1)=v(1)+a-1
412 IF z$(3,2)="C" THEN LET
    v(1)=v(1)+1: LET
    v(3)=v(3)+(v(3)<>26)
414 IF z$(3,2)="R" THEN LET
    v(2)=v(2)+1: LET
    v(4)=v(4)+(v(4)<>26)
```

```
10 POKE 53280,4:POKE 53281,0
20 PRINT"▼◣▣"SPC(16)"▟
```

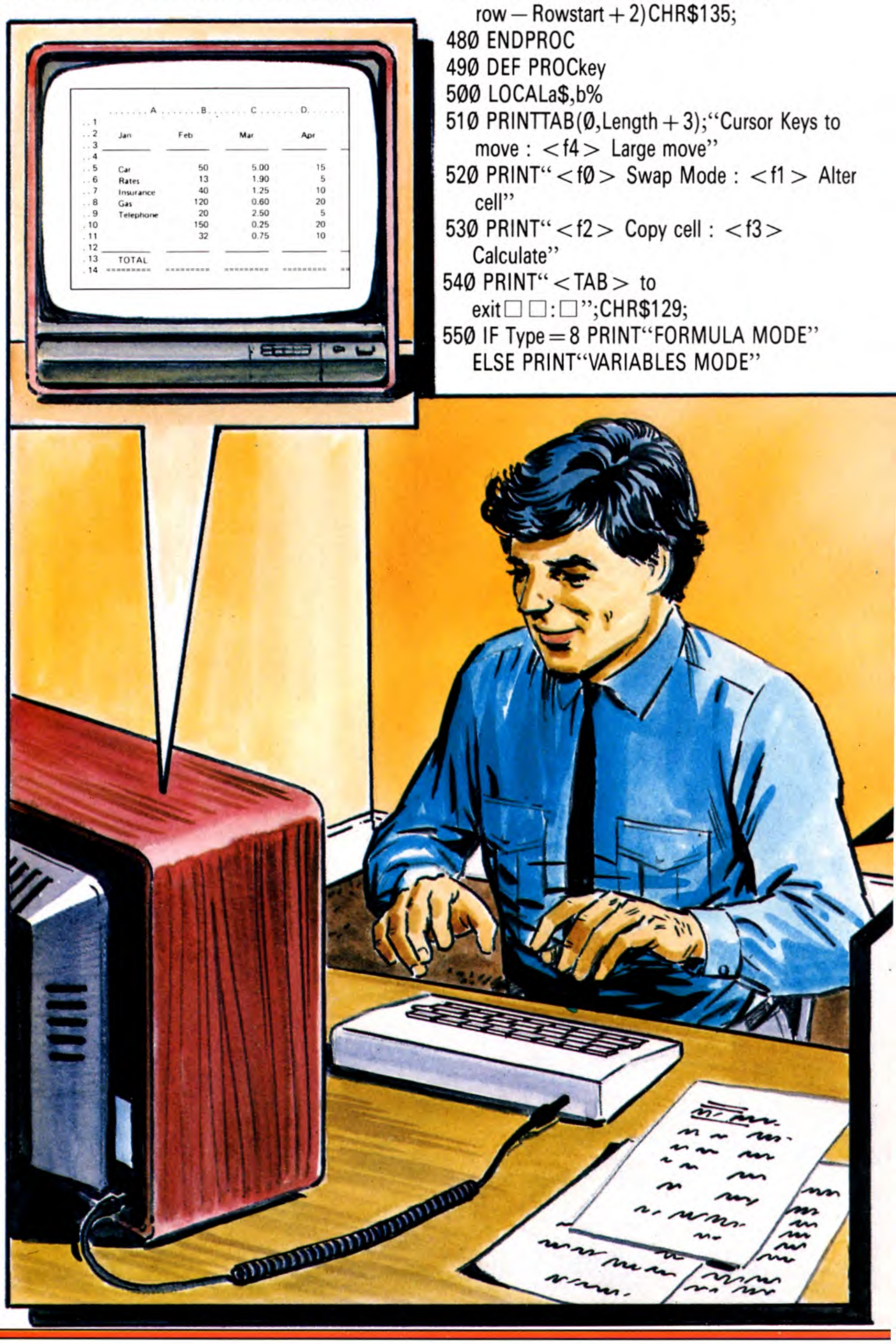
```
   WORKING■"CHR$(8)
30 RM=20:CM=26:LM=15
40 DIM D$(RM,CM)
50 A$=CHR$(128)+"□□□□□□□□
   □□□□□□□□□":AA$=A$:BB$=
   A$:F$=A$
60 FOR R=1 TO RM
70 FOR C=1 TO CM
80 D$(R,C)=A$: NEXT C,R
90 GOSUB 1640
100 C$="■■■■○"+CHR$(133)+
   CHR$(137)+CHR$(134)+CHR$(138)+
   CHR$(135)+CHR$(139)
110 DIM CL(3):CL(0)=5
   :CL(1)=30:CL(2)=158:CL(3)=159
120 OP$="+-*/%$&"
130 RS=1:CS=1:TP=0
140 GOSUB 210
150 IFQ<10 THEN 140
160 GOSUB 1560
170 PRINT"DO YOU WANT TO EXIT
   PROGRAM (Y/N)?"
180 AA$="Y":BB$="N":GOSUB1280
190 IF A$="N" THEN 140
200 POKE 53280,14:POKE 53281,6:PRINT
   "♥◆":END
210 FOR Z=0TO159:POKE 1864+Z,32:
   NEXT Z
220 PRINT"■■■■■■";
230 FOR N=CS TO CS+3
240 PRINT"■■■■";CHR$(64+N);
   "■■■■";
250 NEXT N:PRINT"■";
260 FOR R=RS TO RS+LM-1
270 AA$=STR$(R):IF LEN(AA$)<3 THEN
   AA$="□"+AA$
280 PRINT"■■■"RIGHT$(AA$,2);:N=0
290 FOR C=CS TO CS+3
300 PRINT "■■■"CHR$(CL(N));
310 IF TP=0 THENPRINT RIGHT$(D$(R,C),
   8);
320 IF TP=8 THEN PRINT LEFT$(D$(R,C),8);
330 IF ASC(D$(R,C))=128 AND TP=8
   THEN PRINT "□";
340 N=N+1:NEXT C
350 PRINT "■■■";:NEXT R
360 POKE 198,0:PRINT"■■■■■■■
   ■■■■■■■■■■■■■■
   ■■■■■■■■■■■■■■■
   ■■■■■■■■■■";
370 PRINT"■□CURSOR KEYS TO MOVE□:";
380 IF TP=8 THEN
   PRINT"□■□□FORMULA MODE□■□";
390 IF TP=0 THEN
   PRINT"□π□VARIABLES MODE□■□";
400 PRINT"■□□□□<F1>□SWAP
   MODE□□□:□<F2>□ALTER CELL
   □□";
410 PRINT"■□□□□<F3>□COPY
   CELL□□□:□<F4>□CALCULATE□
   □□";
420 PRINT"■□□□□<F5>□LARGE
   MOVE□□:□<F6>□TO EXIT□□□
   □□■■":PRINT"○";
430 PRINT"""";:GET Q2$:PRINTCHR$(20);:IF
   Q2$="" THEN 430
440 Q1$=C$:GOSUB 2670
450 IF Q=0 THEN 430
460 IF Q=1 THEN CS=CS+1:IF
   CS>CM-3 THEN CS=CM-3
470 IF Q=2 THEN CS=CS-1:IF CS<1
   THEN CS=1
480 IF Q=3 THEN RS=RS-1:IF RS<1
   THEN RS=1
490 IF Q=4 THEN RS=RS+1:IF
   RS>RM-LM+1 THEN
   RS=RM-LM+1
500 IF Q=5 THEN GOSUB560:RETURN
510 IF Q=6 THEN GOSUB680:RETURN
520 IF Q=7 THEN GOSUB1720:RETURN
530 IF Q=8 THEN GOSUB1080:RETURN
540 IF Q=9 THEN GOSUB2590:RETURN
550 RETURN
560 IF TP=0 THEN TP=8:RETURN
570 IF TP=8 THEN TP=0
580 RETURN
590 PRINT"WHICH CELL■?";
600 GET A$:IF A$="" THEN 600
610 IF A$="←" THEN RETURN
620 C=ASC(A$)-64
630 IF C<1 OR C>26 THEN 600
640 PRINT "■"A$
650 INPUT R
660 IF R<1 OR R>20 THEN
```





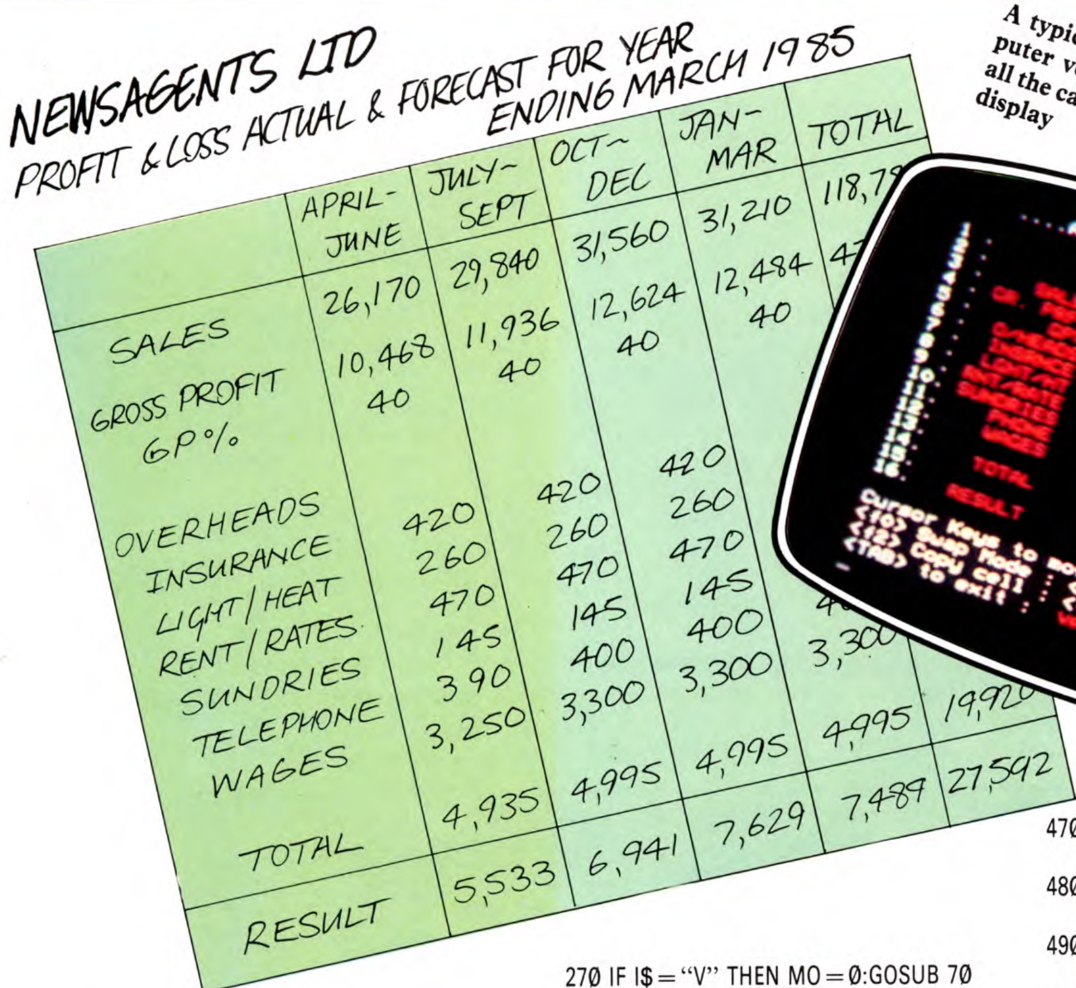
```
   PRINT"◘";:GOTO 65Ø
67Ø RETURN
68Ø GOSUB 59Ø:IF A$="←" THEN RETURN
69Ø PRINT"ENTRY▮";
7ØØ A$="":INPUT A$
71Ø IF TP=8 THEN GOSUB 1Ø4Ø:GOSUB
   75Ø:D$(R,C)=A$+RIGHT$(D$(R,C),8)
72Ø IF TP=Ø THEN GOSUB 75Ø:D$(R,C)=
   LEFT$(D$(R,C),8)+A$
73Ø IF LEFT$(D$(R,C),1)=CHR$(128) THEN
   D$(R,C)="□"+RIGHT$(D$(R,C),15)
74Ø RETURN
75Ø IFLEN(A$)>8THENA$=LEFT$(A$,8)
76Ø IF TP=8 AND LEN(A$)<8 THEN
   A$=A$+"□":GOTO 76Ø
77Ø IF TP=Ø AND LEN(A$)<8 THEN
   A$="□"+A$:GOTO 77Ø
78Ø RETURN
79Ø AA$=MID$(A$,PS,3)
8ØØ BB$=LEFT$(A$,1)
81Ø IF BB$<"A" OR BB$>"W" THEN
   D1=Ø:RETURN
82Ø P=VAL(RIGHT$(AA$,2))
83Ø D1=2:IF P<1Ø THEN D1=1
84Ø IF P>CM OR P<1 THEN D1=Ø
85Ø RETURN
```

```
1Ø MODE7:*FX4,1
2Ø ON ERROR GOTO 3Ø6Ø
3Ø *FX225,14Ø
4Ø Rows=2Ø:Cols=24:Length=15
5Ø DIM D$(Rows,Cols)
6Ø A$=CHR$128+STRING$(15,"□"):
   a$=A$:b$=A$:F$=A$
7Ø FOR r%=1 TO Rows
8Ø FOR c%=1 TO Cols
9Ø D$(r%,c%)=A$
1ØØ NEXT,
11Ø PROCload
12Ø FOR n=136 TO 144:C$=C$+CHR$n:
   NEXT
13Ø C$=C$+CHR$9
14Ø DIM CI(3):CI(Ø)=129:CI(1)=131:CI
   (2)=133:CI(3)=134
15Ø Op$="+-*/%$&"
16Ø Rowstart=1:Colstart=1:Type=Ø
17Ø REPEAT
18Ø PROCmainscreen:PROCkey
19Ø UNTIL K%=1Ø
2ØØ *FX4,Ø
21Ø PROCsave
22Ø PRINT'''"DO YOU WANT TO FINISH
   ?(Y/N)"
23Ø A$=GET$:IF A$="Y" OR A$="y"
   THEN CLS:PRINTTAB(13,1Ø)"Goodbye"
   ''':END
235 *FX4,1
24Ø GOTO17Ø
25Ø DEF PROCmainscreen
26Ø LOCAL r,c,a$,n
27Ø CLS:PRINT"□□□□";
28Ø FOR n=Colstart TO Colstart+3
29Ø PRINT"....";CHR$(64+n);"...□";
3ØØ NEXT
31Ø PRINT
32Ø FOR r=Rowstart TO Rowstart+Length-1
33Ø a$=STR$(r):IF LENa$<2 a$=a$
   +"□"
34Ø PRINTa$;".";:n=Ø
35Ø FOR c=Colstart TO Colstart+3
36Ø PRINTCHR$(CI(n));
37Ø IF Type=Ø PRINTRIGHT$(D$(r,c),8);
38Ø IF Type=8 PRINTLEFT$(D$(r,c),8);
39Ø n=n+1
4ØØ NEXT
41Ø PRINT
42Ø NEXT
43Ø ENDPROC
44Ø DEF PROCflash(row,col)
45Ø IF row<Rowstart OR
   row>Rowstart+Length□ENDPROC
46Ø IF col<Colstart OR col>Colstart+3
   ENDPROC
47Ø PRINTTAB((col-Colstart)*9+3,
   row-Rowstart+2)CHR$135;
48Ø ENDPROC
49Ø DEF PROCkey
5ØØ LOCALa$,b%
51Ø PRINTTAB(Ø,Length+3);"Cursor Keys to
   move : <f4> Large move"
52Ø PRINT"<fØ> Swap Mode : <f1> Alter
   cell"
53Ø PRINT"<f2> Copy cell : <f3>
   Calculate"
54Ø PRINT"<TAB> to
   exit□□:□";CHR$129;
55Ø IF Type=8 PRINT"FORMULA MODE"
   ELSE PRINT"VARIABLES MODE"
```



```
560 REPEAT
570 *FX15,Ø
580 a$=GET$:K%=INSTR(C$,a$):UNTIL K%>Ø
590 IF K%=1 Colstart=Colstart+1:IF Colstart>Cols-3 Colstart=Cols-3
600 IF K%=2 Colstart=Colstart-1:IF Colstart=Ø Colstart=1
610 IF K%=3 Rowstart=Rowstart-1:IF Rowstart<1 Rowstart=1
620 IF K%=4 Rowstart=Rowstart+1:IF Rowstart>Rows-Length+1 Rowstart=Rows-Length+1
630 IF K%=5 PROCswap
640 IF K%=6 PROCalter
650 IF K%=7 PROCreplicate
660 IF K%=8 PROCcalculate
670 IF K%=9 PROCwindowstart
680 ENDPROC
690 DEF PROCswap
700 IF Type=Ø Type=8:ENDPROC
710 IF Type=8 Type=Ø:ENDPROC
720 DEF PROCcellin(vpos)
730 REPEAT
740 INPUTTAB (Ø,vpos)SPC(3Ø)TAB(Ø,vpos)"Which cell□"A$
750 Col=ASC(A$)-64:Row=VAL(MID$(A$,2))
760 UNTIL (Row>=Rowstart AND Row<Rowstart+Length) AND (Col>=Colstart AND Col<Colstart+4)
770 PROCflash(Row,Col)
780 ENDPROC
790 DEF PROCalter
800 LOCAL vpos
810 vpos=VPOS
820 PROCcellin(vpos)
830 INPUT TAB(14,vpos)"□Entry□"A$
840 IF Type=8 PROCformulacheck:D$(Row,Col)=FNformat+RIGHT$(D$(Row,Col),8)
850 IF Type=Ø D$(Row,Col)=LEFT$(D$(Row,Col),8)+FNformat:IF LEFT$(D$(Row,Col),1)=CHR$128 D$(Row,Col)="□"+RIGHT$(D$(Row,Col),15)
860 ENDPROC
```

```
10 PMODEØ,1:PCLEAR1:CLEAR 1ØØØØ:CLS:PRINT@23Ø,"SPREADSHEET PROGRAM"
20 CS=1:RS=1:CR=1:CC=1:MO$(Ø)="VALUE (CALC)":MO$(1)="EQUATION□□□□":MO=1:OP$="+-*/%$&"
30 DIM D$(26,3Ø),D(26,3Ø)
40 FOR I=1 TO 26:FOR J=1 TO 3Ø:D$(I,J)=CHR$(128):NEXT J,I
50 CX=4:RX=1
60 GOSUB 7Ø:GOTO 17Ø
70 PRINT@448,"WAIT":PRINT@Ø,STRING$(3,128);:FOR I=CS TO CS+3:PRINT CHR$(123);CHR$(128);CHR$(128);CHR$(96+I);CHR$(128);CHR$(128);CHR$(125);:NEXT:PRINTCHR$(128);
80 PRINT@48Ø,"MODE:□";MO$(MO);
90 FOR I=Ø TO 11:C1=INT((RS+I)/1Ø)+48:C2=(RS+I)-((C1-48)*1Ø)+48:POKE 1Ø24+32*I+32,C1:POKE 1Ø24+32*I+33,C2:PRINT@32*I+34,"":NEXT
100 PRINT@416:IF MO=Ø THEN GOSUB 74Ø:GOTO 13Ø
110 FOR J=RS TO RS+11:FOR I=CS TO CS+3
120 PRINT@(J-RS)*32+35+(I-CS)*7,"";:GOSUB 66Ø:NEXT I,J
130 PRINT@48Ø,"MODE:□";MO$(MO);TAB(2Ø);"CELL:□";CHR$(64+CC);MID$(STR$(CR),2);"□";
```



A typical 'paper' system, and the computer version. The computer performs all the calculations as well as holding the display

```
140 PRINT@448,“READY”
150 PRINT@458,MID$(D$(CC,CR),2)
160 RETURN
170 PS=(CR−RS+1)*32+(CC−CS)
    *7+3+1Ø24:Z=PEEK(PS):POKE PS,191
    ANDZ
180 I$=INKEY$:IF I$=“” THEN 18Ø
190 POKE PS,Z
2ØØ IF I$=CHR$(8) AND CC>1 THEN
    CC=CC−1:IF CC<CS THEN
    CS=CS−1: GOSUB 7Ø
21Ø IF I$=CHR$(9) AND CC<26 THEN
    CC=CC+1:IF CC>CS+3 THEN
    CS=CS+1:GOSUB 7Ø
22Ø IF I$=CHR$(1Ø) AND CR<3Ø THEN
    CR=CR+1:IF CR>RS+11 THEN
    RS=RS+1:GOSUB 7Ø
23Ø IF I$=CHR$(94) AND CR>1 THEN
    CR=CR−1:IF CR<RS THEN
    RS=RS−1:GOSUB 7Ø
24Ø IF I$=G” GOSUB 33Ø
25Ø IF I$=“Q” THEN CLS:INPUT “ARE YOU
    SURE YOU WANT TO QUIT□□□
    (Y/N)”; A$:IF A$<>“Y” THEN GOSUB
    7Ø ELSE CLS:END
26Ø IF I$=“I” GOSUB 41Ø
27Ø IF I$=“V” THEN MO=Ø:GOSUB 7Ø
28Ø IF I$=“C” GOSUB 149Ø
29Ø IF I$=“E” THEN MO=1:GOSUB 7Ø
3ØØ IF I$=“S” GOSUB 123Ø
31Ø IF I$=“L” GOSUB 135Ø
32Ø GOSUB 13Ø:GOTO 17Ø
33Ø PRINT@448:PRINT@448,“GOTO
    CELL >”;:LINE INPUT A$
34Ø IF A$=“” THEN RETURN
35Ø C1=ASC(A$)− 64: IF C1<1 OR
    C1>26 THEN 33Ø
36Ø C2=VAL (MID$(A$,2)):IF C2<1 OR
    C2>3Ø THEN 33Ø
37Ø CC=C1:CS=C1:CR=C2:RS=C2
38Ø IF CS>23 THEN CS=23
39Ø IF RS>19 THEN RS=19
4ØØ GOSUB 7Ø:RETURN
41Ø PRINT@448,“ENTER NEW
    CONTENTS:”;:LINE INPUT A$
42Ø IF A$=“” THEN A$=CHR$(128):GOTO
    61Ø
43Ø IF LEN(A$)>9 THEN
    PRINT@448,“INVALID ENTRY”:SOUND
    1,4:GOTO 41Ø
44Ø IF VAL(A$)<>Ø THEN 56Ø
45Ø B$=LEFT$(A$,1):IF B$<“A” OR
    B$>“Z” THEN 6ØØ
46Ø C$=MID$(A$,2,2)
47Ø IF VAL(C$)<1 OR VAL(C$)>3Ø THEN
    6ØØ
48Ø IF VAL(C$)<1Ø THEN A$=B$+STR$
    (VAL(C$))+MID$(A$,3)
49Ø D$=MID$(A$,4,1):IF D$<“A” OR
    D$>“Z” THEN 6ØØ
5ØØ E$=MID$(A$,5)
51Ø IF VAL(E$)<1 OR VAL (E$)>3Ø THEN
    6ØØ
52Ø IF VAL(E$)<1Ø THEN A$=
    LEFT$(A$,4)+STR$(VAL(E$))+MID$
    (A$,6)
53Ø O$=MID$(A$,7,1):IF INSTR(1,
    OP$,O$)=ØORO$=“” THEN 6ØØ
54Ø DP=VAL(RIGHT$(A$,1)):IF DP<Ø OR
    DP>7 THEN 6ØØ
55Ø PRINT@448,“ENTRY IS AN equation”:
    A$=CHR$(131)+A$:GOTO 61Ø
56Ø PRINT@448,“ENTRY IS A value”
57Ø IF RIGHT$(A$,1)=“□” THEN A$=
    LEFT$(A$,LEN(A$)−1):GOTO 57Ø
58Ø IF LEN(A$)<7 THEN A$=“□”+A$:
    GOTO58Ø
59Ø A$=CHR$(129)+A$:GOTO 61Ø
6ØØ PRINT@448,“ENTRY IS A label”:
    A$=CHR$(13Ø)+A$
61Ø D$(CC,CR)=A$:I=CC:J=CR:PRINT@
    (J−RS)*32+35+(I−CS)*7,“”;:GOSUB
    66Ø:SOUND19Ø,2:FORD=1TO5ØØ:NEXT
62Ø IF CC>CX THEN CX=CC
63Ø IF CR>RX THEN RX=CR
64Ø IF MO=Ø THEN MO=1:GOSUB 7Ø
65Ø RETURN
```

CLIFFHANGER: RESETTING VARIABLES

The 'CLIFFHANGER' listings published in this magazine and subsequent parts bear absolutely no resemblance to, and are in no way associated with, the computer game called 'CLIFF HANGER' released for the Commodore 64 and published by New Generation Software Limited.

Getting everything into the right place at the right time is one of the most complicated parts of game construction. Here are the routines to get things happening in sync

Each time a game of Cliffhanger begins, it is not just the score that has to be set. You also have to tell other routines how you want them to start off. You have to drain the sea back to the bottom of the screen, tell the cloud which way the wind is blowing, make Willie stand still on the starting line and set various delays so that everything happens in the correct sequence on the screen.

The following routine sets a series of variables to the values they need to carry when Willie starts out on his hazardous task:

```
ORG  58606
DTH  ld a,6
     ld (57353),a
     ld hl,736
     ld (57354),hl
     ld hl,130
     ld (57345),hl
     ld a,3
     ld (57347),a
     ld a,0
     ld (57348),a
     ld a,2
     ld (57349),a
     ld hl,449
     ld (57332),hl
     ld hl,0
     ld (57334),hl
     ld a,0
     ld (57336),a
     ld hl,223
     ld (57356),hl
     ld a,0
     ld b,5
     ld (57350),a
     add a,b
     ld (57351),a
     add a,b
     ld (57352),a
```

This routine is labelled dth because it is called not just at the beginning of the game, but after

Willie's death too. It sets the game up again for another go.

SETTING THE SEA

There is a great advantage in having all your variables together in one place. It allows you to check exactly what state the game is in at any time while you are debugging it.

One-byte variables are set via the eight-bit accumulator while two-byte variables are set via the 16-bit HL register pair, even if the amount being loaded at this stage can be contained in one byte. This is because the high bytes of the variable must be set too.

Memory location 57,353 contains the sea delay. This is loaded with 6 to give Willie a reasonable chance to scale the cliff before he gets drowned. Later the delay can be changed to speed up the onrush of the sea and make the game more difficult and more exciting.

The sea must also start at the bottom of the screen at the beginning of each screen. The screen position of the left-hand end of the top of the sea is stored in 57,354 and 57,355. The number 736 is loaded in there which is the screen position at the bottom left-hand corner of the screen.

CLOUDING THE ISSUE

In the Spectrum version of Cliffhanger the cloud moves about. Memory location 57,345 is its screen position and this is loaded with 13∅, the position it should start from.

But that's not all the game needs to know about the cloud. It needs a delay so that the cloud does not zoom around like an aircraft. The delay variable is stored in 57,347 and this is loaded with 3 to set it.

The cloud also needs to know which direction it is travelling in. This information is stored in 57,348. A ∅ in this location means that the cloud is moving to the right. A 1 means that it is moving to the left. Here you initialize the routine by storing ∅ in this location, sending the cloud to the right.

FLY, STAND, DIE

The gull delay is stored in 57,349 and this is set to 2. Willie's screen position is given by the contents of 57,332, so this is loaded with 449 which is the screen position of the bottom left-hand end of the slope.

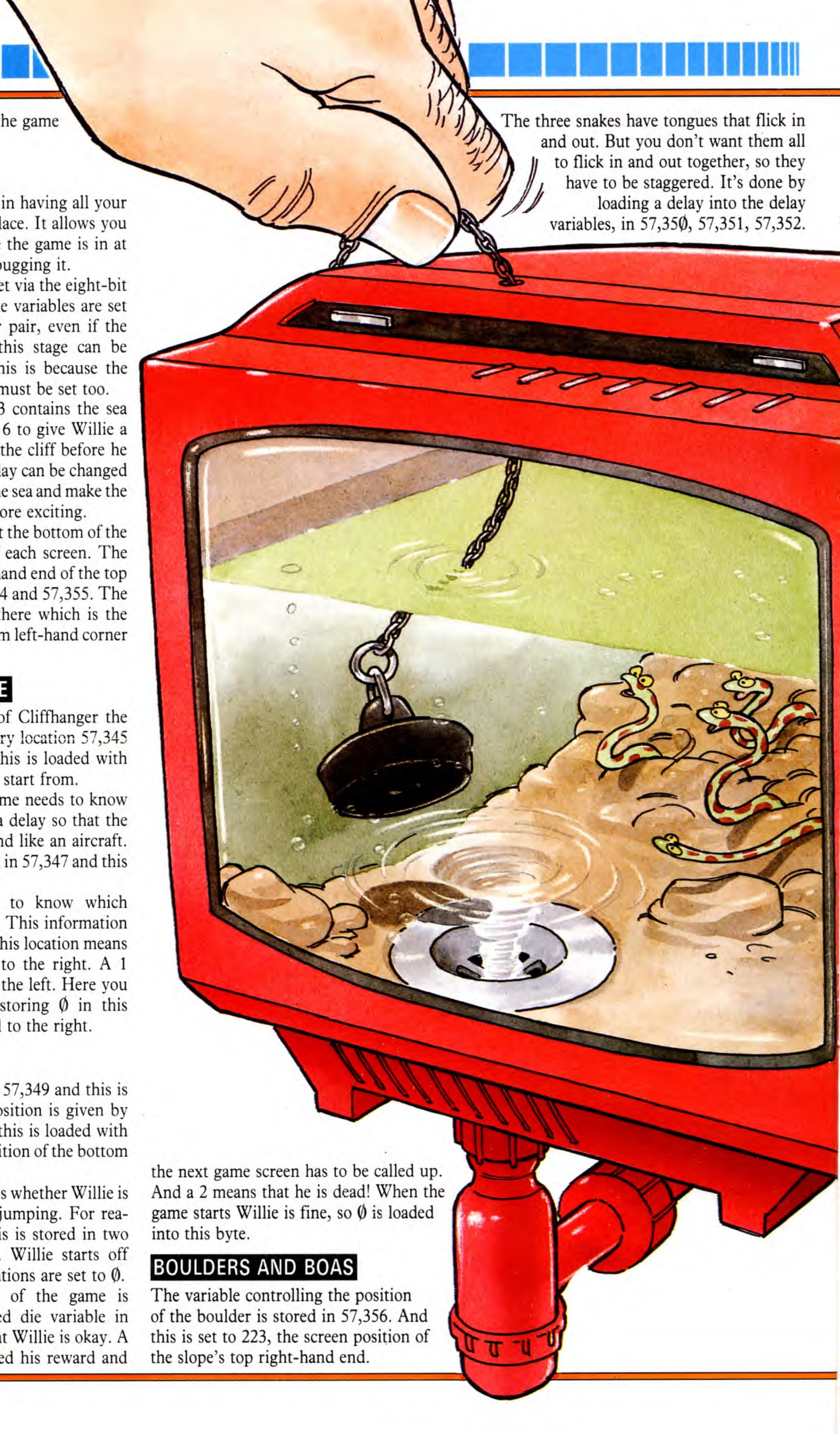
Another variable controls whether Willie is standing still, running or jumping. For reasons you will see later, this is stored in two bytes, 57,334 and 57,335. Willie starts off standing still, so these locations are set to ∅.

The general condition of the game is monitored by the so-called die variable in 57,336. A ∅ here means that Willie is okay. A 1 means that he has reached his reward and the next game screen has to be called up. And a 2 means that he is dead! When the game starts Willie is fine, so ∅ is loaded into this byte.

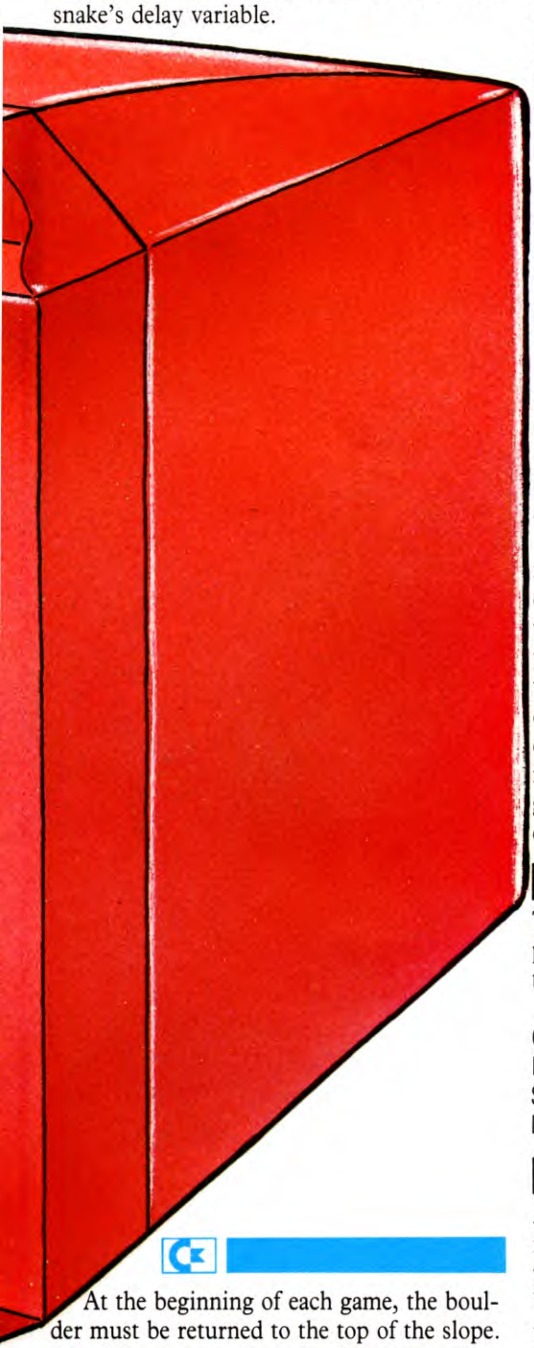
BOULDERS AND BOAS

The variable controlling the position of the boulder is stored in 57,356. And this is set to 223, the screen position of the slope's top right-hand end.

The three snakes have tongues that flick in and out. But you don't want them all to flick in and out together, so they have to be staggered. It's done by loading a delay into the delay variables, in 57,35∅, 57,351, 57,352.



A is loaded with 0 and B with the stagger, 5. The 0 is stored in the first snake's delay variable. The 5 in B is added to the 0 in A and the result, 5, is stored in the second snake's delay variable. Another 5 from B is added to that 5 and the result, 10, is stored in the third snake's delay variable.



At the beginning of each game, the boulder must be returned to the top of the slope. The following routine does that:

```
ORG  22608
LDA  #56
STA  $D002
LDA  $D010
ORA  #2
STA  $D010
LDA  #81
STA  $D003
LDA  #72
STA  $C008
LDA  #13
STA  $C009
RTS
```

ROCK ON

The boulder is sprite one and its X and Y coordinates are held in memory locations $D002 and $D003 on the Vic chip. But because the Commodore's screen is 320 screen positions wide— and only numbers up to 255 can be accommodated in one memory location—a further memory location must be used to hold the most significant bit. The MSB register is memory location $D003.

The boulder starts its roll from X position 312 and Y position 81. So 56 is loaded into the accumulator and stored in memory location $D002. And the MSB register at $D003 is set by ORing its contents with 2. Then the accumulator is loaded with 81 which is stored in memory location $D002.

But that is not the end of the story. It is no good having the boulder leaping from one screen position to the next. Convincing animation depends on smooth action. And to achieve that, the boulder must be moved half a screen position at a time.

Within this program the half position movement is done by using what are known as double density coordinates. How these work will be seen in a later part of Cliffhanger when you come to move the boulder. For now through hough you have to set the double density X coordinate to 72 and the double density Y coordinate to 13. These are stored in memory locations $C008 and $C009 in the game's variable table and initialize the boulder sprite to its start position.

CLOUDS

The cloud must be set to its correct start position too, and it must start off travelling in the right direction:

```
ORG  24912
LDA  #1
STA  $C00B
LDA  #50
STA  $D004
LDA  #70
STA  $D005
RTS
```

WHICH WAY BLOWS THE WIND?

Memory location $C00B in the variable table is used as a flag to tell the cloud which way the wind is blowing. A 0 means that the wind is blowing from east to west and the cloud is travelling to the left and a 1 means that the wind is blowing west to east and the cloud is moving to the right. To start off with the cloud should move to the right, so 1 is loaded into the accumulator and stored in 49,163.

The cloud is sprite number two, whose X and Y coordinates are stored in memory locations $D004 and $D005. The initial cloud position is X = 50 and Y = 70. So 50 is loaded into the accumulator and stored in $D004. And 70 is loaded into the accumulator and stored in $D005.

The cloud is not going to move to any position further right than 255, so the MSB register does not have to be set.

This program sets up all the variables in the zero page at the beginning of each game. And it resets all of them—with the exception of the score and the lives left—at the beginning of each screen. Don't forget to set PAGE = &3000 and type NEW and *TAPE before you key it in.

```
110 DATA0,0,1,38
120 DATA46,20,14,0
130 DATA0,0,0,4
140 DATA0,0,0,10
150 DATA0,10,0,0
160 DATA5,0,0,0
170 DATA0,0,0
180 FORA% = &1D5CTO&1D76:READ?A%:
    NEXT
190 FOR PASS = 0TO3STEP3
220 P% = &1D77
230 [OPTPASS
240 .Init
250 LDX#0
260 .Lb1
270 LDA&1D5C,X
280 STA&75,X
290 INX
300 CPX#27
310 BNELb1
320 JSR&1BA3
330 RTS
340 ]
430 DATA0,0,1,38
440 DATA46,20,14,0
450 DATA0,0,0,4
460 DATA0,0,255,255
470 DATA0,10,0,0
480 IFPASS = 0
     THEN FORA% =
     &1D87TO&1D9A:
     READ?A%:NEXT
520 P% = &1D9B
530 [OPTPASS
540 .InitSc
550 JSR&1B32
560 LDX#0
570 .Lb2
580 LDA&1D87,X
590 CMP#255
600 BEQLb3
610 STA&75,X
620 .Lb3
630 INX
640 CPX#20
650 BNELb2
660 LDY#0
670 .Lb4
680 LDA#1
690 STA&77
700 JSR&1CCB
710 INY
720 CPY#8
730 BNELb4
740 RTS
750 ]NEXT
```

You will notice that this routine jumps a couple of other subroutines that have not been published so far. So if you call it, it will crash.

The way to get round this is to POKE RTSs (96) into the start addresses of the subroutines it jumps to after you have *SAVED the machine code and assembly language. This will send the processor back straight away.

The locations in question are &1BA3 and &1CCB. POKE these with 96 which is the code for RTS.

When you have done that, and have the rest of the game in memory, call the routine with this instruction:

```
CALL &1D77
```

Nothing should happen. In fact, all the routine at that address does is to set up the variables, so you'll see no effect on the screen. To find out whether it has worked or not, try:

```
CALL &1D9B
```

But before that you must put RTSs (96) in at memory locations &IAZE and &IA3C. This should print up the first screen with the score set to zero, the lives to five and the level to 1 which is a screen with potholes.

NEW GAME

The DATA in Lines 11Ø to 17Ø is the initialization values of all the variables in the game. When the program is RUN, the BASIC instructions in Line 18Ø POKE it into an initialization table at &1D5C to &1D76. The following machine code routine picks those initialization values up one at a time from the initialization table and copies them into the variables table.

It may seem unnecessary to have this data in more than one place. But when this program has been RUN and the BASIC is removed, the initialization table will be the only constant source of reference for these initialization values. The values of the variables in the variables table are updated throughout the game and the only way to set them back to what they were at the beginning is to copy their values out of the initialization table again. The machine-code program does this.

LDX #Ø sets the offset in the X register to Ø and LDA &1D5C,X loads the first byte of the initialization table into the accumulator. STA &75,X stores it in the first location of the variables table. X is then incremented to move the LDA instruction onto the next byte of the initialization table and the STA instruction onto the next location in the variables table.

The processor goes round and round the Lb1 loop, loading up the next byte of the initialization table and storing it in the next location in the variables table until all 27 of the variables have been initialized. When X has clocked up to 27, the CPX #27 instruction sets the zero flag, the condition of the BNE instruction is no longer fulfilled and the processor drops out of the loop.

The routine then sends the processor to the subroutine at &1BA3. This routine sets the sound envelope to make the tune and the sound effects quiet. But it is not in position at the moment and this is one of the locations that you should have POKEd an RTS.

When the processor returns, it hits another RTS in this program and returns to the place this routine was called from.

NEW SCREEN

The DATA in Lines 43Ø to 47Ø is the data required to reinitialize a new screen. This DATA is READ into a second table at &1D87 to &1D9A. You will note that it is very like the first 20 bytes of the DATA given for a new game. The last seven bytes deal with lives and the score and so do not have to be reinitialized at the beginning of the screen.

There are a couple of other variables that do not have to be reset either. You will notice that the Ø and the 1Ø at the end of Line 14Ø have been replaced with 255s in Line 46Ø. These variables are not going to be reset either—you'll see why in the machine code programming.

The instruction on Line 55Ø sends the processor off the subroutine which prints the screen up. Then X is set to Ø again and another loop is executed which copies the initialization values from the second data table into the variables' locations.

But this time, between the load and store instructions on Lines 58Ø and 61Ø, the byte of data is compared with 255. And if it is 255, the BEQ instruction skips the STA. So the two 255s are not stored in the appropriate variable locations and the values in those locations are carried forward unchanged from screen to screen.

When the processor has finished initializing the 18 variables that need to be reset between screens, it goes on to print up the first line of the sea.

The variable in &77 is the so-called sea delay. This is a counter which is counted down between each advance of the sea. It's a simple device to stop the sea filling up the screen too fast.

Normally, during the game, after each advance of the tide, it is set to 5 and is counted down to zero again before the next advance is made. Here, though, it is loaded with 1—so when the sea routine at &1CCB is called, it decrements the counter to Ø and the first pixel line of the sea is printed up.

The counter in Y is set to 0 in Line 660 at the beginning of this subroutine. It is incremented in Line 710, compared with 8 in Line 720 and tested in Line 730. So the processor goes round this loop 8 times. Each time the loop is executed the sea delay is set back to 1, so each time the sea routine is called it prints up another pixel line of sea. It's called eight times, so the first character line of sea is printed up on the screen.

Unfortunately, you do not have this sea routine yet so the JSR will simply return without any effect—if you have POKEd 96 into &1CCB.

When the sea routine has been called eight times, the processor drops out of the routine and returns.

The following routine sets a series of variables to the values they need to carry when Willie starts out on his hazardous task:

```
     ORG 19447
NLV  LDA #6
     STA 18246
     LDX #7424
     STX 18247
     LDX #5088
     STX 18249
     CLR 18251
     CLR 18252
     LDX #3070
     STX 18253
     CLR 18255
     LDA #5
     STA 18256
     LDA #10
     STA 18257
     RTS
```

This routine is labelled NLV (or New LiVe) because it is called not just at the beginning of the game, but after Willie's death.

SEA SET

There is a great advantage in having all your variables together in one place. It allows you to check exactly what state the game is in at any time while you are debugging it.

One-byte variables are set via the eight-bit accumulator, while two-byte variables are set via the 16-bit X register, even if the amount being loaded at this stage can be contained in one byte. This is because the high byte of the variable must be set too.

Memory location 18,246 contains the sea delay. This is loaded with 6 to give Willie a reasonable chance to scale the cliff before he gets drowned. Later, the delay can be changed to speed up the onrush of the sea and make the game more difficult and more exciting.

The sea must also start at the bottom of the screen at the beginning of each screen. The screen position of the left-hand end of the top of the sea is stored in 18,247 and 18,248. The number 7,424 is loaded in there which is the screen position at the bottom left-hand corner of the screen.

LIFE AND DEATH

Willie's screen position is given by the contents of 18,249, so this is loaded with 5,088 which is the screen position of the bottom left-hand end of the slope which is where Willie starts off from.

Another variable, in 18,251, controls whether Willie standing still or running and jumping. A 0 here gives the first UDG picture of Willie, that is Willie standing still.

The general condition of the game is monitored by the so-called die variable in 18,252. A 0 here means that Willie is okay. A 1 means that he has reached his reward and the next game screen has to be called up. And a 2 means that he is dead! But when the game starts off Willie is okay, so this byte is cleared.

STONES AND SNAKES

The variable controlling the position of the boulder is stored in 18,253. And this is set to 3,070, the screen position of the top right-hand end of the slope where the boulder begins its roll.

The three snakes have tongues that flick in and out. But you don't want them all to flick in and out together, so they have to be staggered. This is done by loading a delay into the three delay variables in 18,255, 18,256 and 18,257.

The first snake's delay variable is set to 0 by clearing it. Five is stored in the second snake's delay variable. And 10 is stored in the third snake's delay variable.

MORE ABOUT PAGED GRAPHICS

Further investigation of paged graphics gives more insights into the usefulness of this technique. Here are more programs to demonstrate them on your micro.

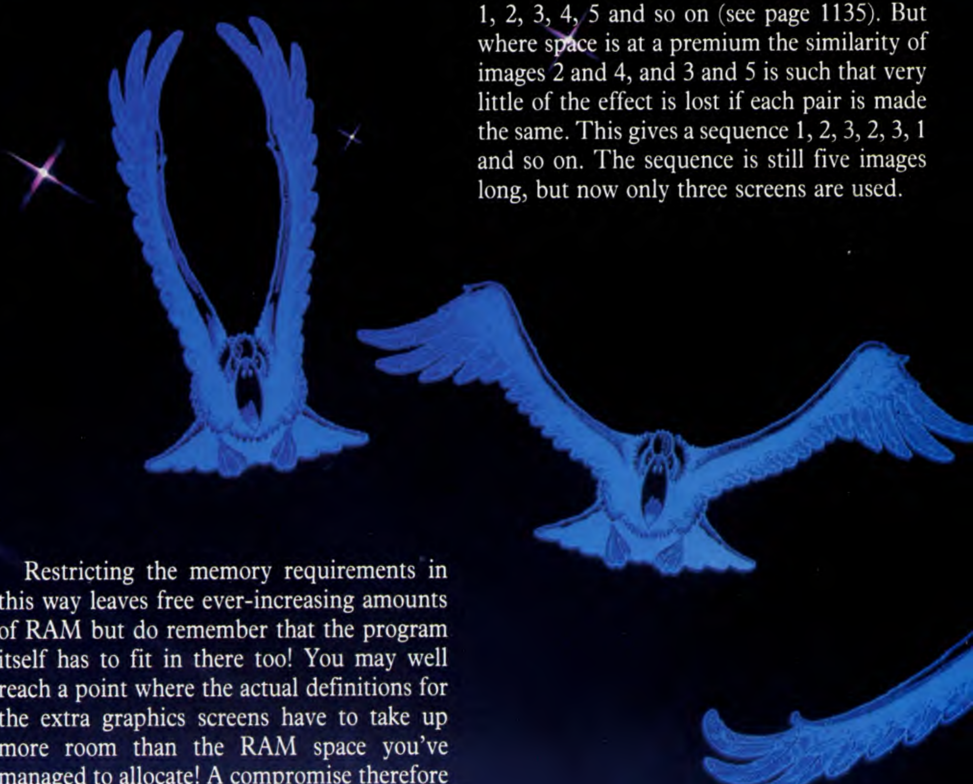
Paged graphics—the technique of flipping from one graphics screen to another—offers considerable potential in many different types of application where a fast change over from one screenful of data to another is desired. Although an obvious use is in computerized animation, paged graphics can of course be put to rather more serious uses, an example of which is graphs, or illustrating separate screens of figures, such as may be used in various types of financial program.

You have already seen one example of the technique, on pages 1022 to 1028. Now it's time to explore a little further. To recap briefly, the principle behind paged graphics is to define and then confine to memory data for entirely separate screens. This data can take the form of high or low resolution graphics, even text—perhaps a combination. Each of these screens of data can be called up in turn, in very quick succession, without needing the characteristic 'building time' between each screen normal for a graphics display.

This building time is still required, but needs only take place once for each screen—before the main display starts—and then this screen is confined to a suitable area of memory from where it can be recalled almost instantly as required.

MEMORY RESTRICTIONS

Each 'page' of screen data requires a certain amount of memory. How much memory you need varies, because the more colours you use and the higher the resolution of the graphics, the amount of memory required for each screen is greater. There are in any case severe memory restrictions on some home computers and the only way to employ paged graphics on these is to restrict each screen of graphics to a fraction of the normal depth—a third or less perhaps. Also, it's often necessary to sacrifice colours and resolution.



Restricting the memory requirements in this way leaves free ever-increasing amounts of RAM but do remember that the program itself has to fit in there too! You may well reach a point where the actual definitions for the extra graphics screens have to take up more room than the RAM space you've managed to allocate! A compromise therefore has to be established—and for each machine this can be translated into practicable limits for the numbers of screen pages available.

The paging technique can call individual screens from memory in any order and more than once in any sequence if desired. So it's quite possible to construct a paging *sequence* of perhaps eight screens although there are very much fewer screens in memory. This useful memory saving technique is especially effective if care is taken to ensure that the graphics of the repeated intermediate screens do nothing to detract from the 'flow' of the animation. Thus, as an example, in a sequence of paged graphics depicting a stick man walking you could well have pages 1, 2, 3, 4, 5, 1, 2, 3, 4, 5 and so on (see page 1135). But where space is at a premium the similarity of images 2 and 4, and 3 and 5 is such that very little of the effect is lost if each pair is made the same. This gives a sequence 1, 2, 3, 2, 3, 1 and so on. The sequence is still five images long, but now only three screens are used.

The Spectrum 48K can, at the most, handle eight or nine separate screen pages but only in two (INK and PAPER) colours. You are also limited to the amount of screen available. This is based on using about two-thirds screen depth which accounts for 4K per screen. Add another 2K or so for the program itself and the practical limit does appear to be eight pages, and this is what the following program—'roadway perspective'—is based on.

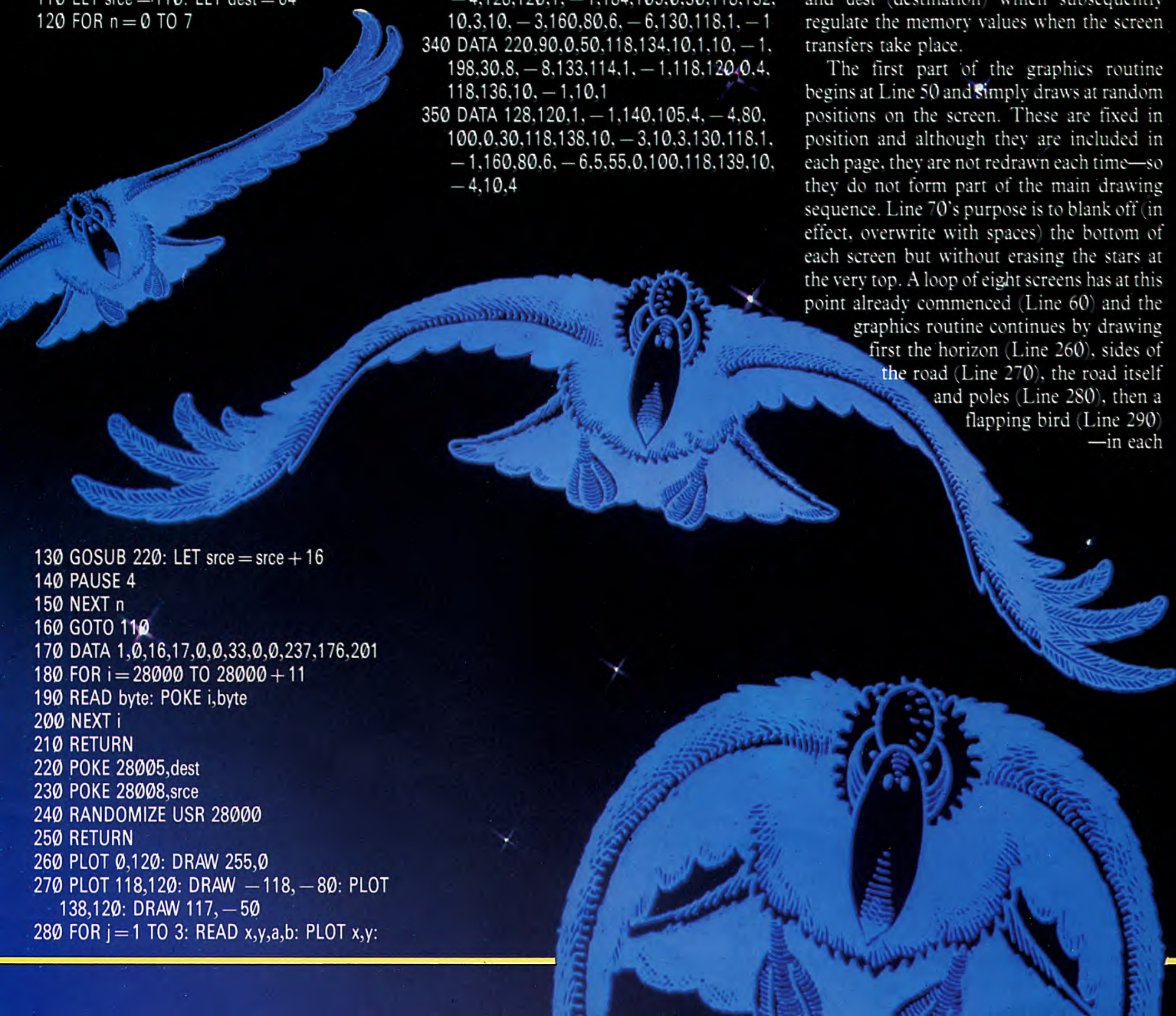
```
10 BORDER 0: PAPER 0: INK 7: CLS
20 CLEAR 27999
30 GOSUB 170
40 LET srce=64: LET dest=110
```

- PAGED GRAPHICS RECAP
- LIMITS TO MEMORY
- THE PAGES AVAILABLE ON YOUR MACHINE
- PAGING TECHNIQUE
- PAGED GRAPHICS AND SPACE REQUIREMENTS
- STORING AND RETRIEVING
- USING FEWER PAGES FOR ANIMATION EFFECTS

```
50 FOR n=1 TO 20: PLOTRND*(255),RND*
   (40)+130: NEXT n
60 FOR n=Ø TO 7
70 FOR m=4 TO 21: PRINT AT m,0:"□□
   □□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□
   □": NEXT m
80 GOSUB 260
90 GOSUB 220: LET dest=dest+16
100 NEXT n
110 LET srce=110: LET dest=64
120 FOR n=0 TO 7
130 GOSUB 220: LET srce=srce+16
140 PAUSE 4
150 NEXT n
160 GOTO 11Ø
170 DATA 1,Ø,16,17,Ø,Ø,33,Ø,Ø,237,176,201
180 FOR i=28000 TO 28000+11
190 READ byte: POKE i,byte
200 NEXT i
210 RETURN
220 POKE 28005,dest
230 POKE 28008,srce
240 RANDOMIZE USR 28000
250 RETURN
260 PLOT Ø,120: DRAW 255,Ø
270 PLOT 118,120: DRAW -118,-80: PLOT
    138,120: DRAW 117,-5Ø
280 FOR j=1 TO 3: READ x,y,a,b: PLOT x,y:
    DRAW a,b: NEXT j
290 READ x,y,a,b,c,d
300 PLOT x,y: DRAW a,b: DRAW c,d
310 RETURN
320 DATA 128,120,1,-1,140,105,3,-3,138,
    120,0,2,118,140,10,-5,10,5,130,118,1,
    -1,160,80,6,-6,143,118,0,7,118,138,
    10,-3,10,3
330 DATA 133,114,1,-1,198,30,8,-8,160,
    112,0,15,118,136,10,-1,10,1,140,105,4,
    -4,128,120,1,-1,184,105,0,30,118,132,
    10,3,10,-3,160,80,6,-6,130,118,1,-1
340 DATA 220,90,0,50,118,134,10,1,10,-1,
    198,30,8,-8,133,114,1,-1,118,120,0,4,
    118,136,10,-1,10,1
350 DATA 128,120,1,-1,140,105,4,-4,80,
    100,0,30,118,138,10,-3,10,3,130,118,1,
    -1,160,80,6,-6,5,55,0,100,118,139,10,
    -4,10,4
```

The program starts by setting the screen colour to black and then RAMTOP to 27999. A small machine-code routine is then placed above the cleared RAM space by the routine in Lines 170, 180, 190 200 and 210. The purpose of this is to handle the transfer of the screen data 'blocks' to memory as they're created and, later, as they're recalled from memory for display. Line 40 sets the initial high byte values of the variable srce (source) and dest (destination) which subsequently regulate the memory values when the screen transfers take place.

The first part of the graphics routine begins at Line 50 and simply draws at random positions on the screen. These are fixed in position and although they are included in each page, they are not redrawn each time—so they do not form part of the main drawing sequence. Line 70's purpose is to blank off (in effect, overwrite with spaces) the bottom of each screen but without erasing the stars at the very top. A loop of eight screens has at this point already commenced (Line 60) and the graphics routine continues by drawing first the horizon (Line 260), sides of the road (Line 270), the road itself and poles (Line 280), then a flapping bird (Line 290) —in each



instance READing data from the block at the end of the program (Lines 32Ø onwards).

RETURNing from the drawing routine, the program goes to a POKE routine beginning at Line 24Ø to copy the 4K screen into its appropriate place in memory. The dest address is now incremented by a high byte value of 16 (16*255 = 4K) to create the 4K storage space needed for the next page of graphics. The program then loops through the drawing routine again, and the whole cycle is repeated for a total of eight times, with a slightly different 'frame' being created on each pass.

The program then successively calls up the page blocks to create the animated paged graphics sequence. In effect, the 'dest' locations become the new 'srce' locations and are in turn called from memory using the POKE and USR routine in Lines 22Ø to 25Ø.

You can use the main routine without modification to create your own paged graphics. Just replace the drawing sequences (Lines 5Ø, 7Ø, 8Ø and Lines 26Ø onwards) with your own graphics routines. If you do not use fixed elements like the stars, you will not need a Line 5Ø outside the drawing loop, and Line 7Ø can be replaced by a CLS as none of the screen need be preserved. Don't make your program too long, as it must not use memory that is required for the pages.

Only two or three full-area hi-res screens can be retained in memory at once. Bit-mapping the whole screen into memory means that each would require about 8K of memory. Obviously, smaller sections of the screen area may be used to increase the number of pages available for paged graphics and this is what's been done in the example that follows to give five screen pages.

This program makes use of Simons' BASIC or, with amendments, the *INPUT* equivalent (the high-res facility). Each system imposes certain memory restrictions and only approximately 16K is left available for screen page memory. For instance, the Simons' BASIC extension makes use of RAM between 8192 and 16384 and it's convenient to make use of RAM above this for the graphics pages.

```
2Ø POKE 51,255:POKE 52,29:POKE 55,
   255:POKE 56,29:CLR
3Ø GOSUB 22Ø
4Ø D=64
5Ø FOR N=Ø TO 4
6Ø HIRESØ,1:MULTI 7,4,3:COLOUR 6,Ø
```

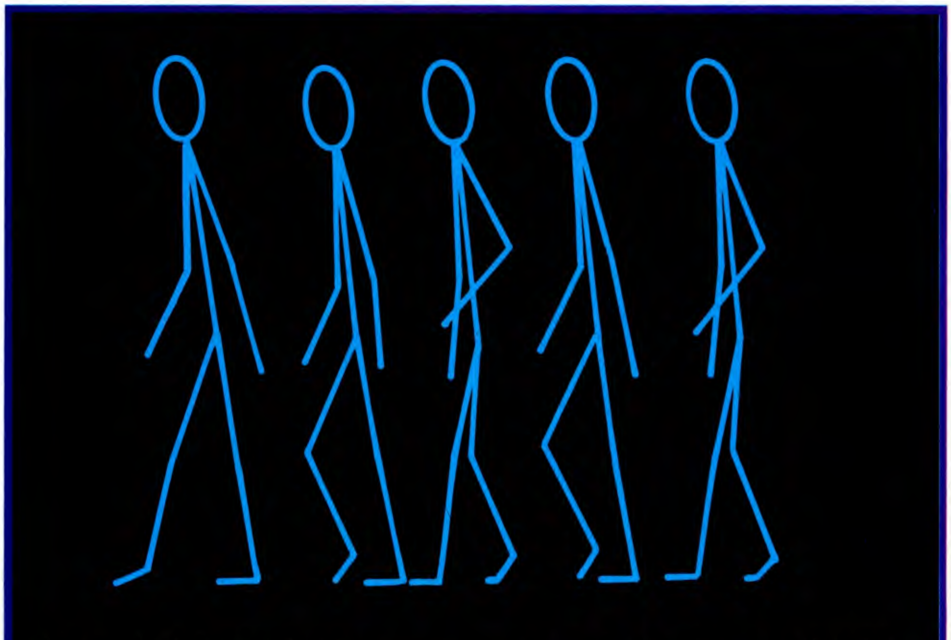
```
70 FOR Z=1 TO 12:FOR ZZ=1 TO 3:LINE
   Z*Z+ZZ*(N+1),0,Z*Z+ZZ*(N+1),100,
   ZZ
75 PLOT RND(1)*160,RND(1)*100,RND(1)*
   3+1:NEXT ZZ,Z
80 FOR ZZ=1 TO 3:LINE 0,(N*13)+ZZ*
   (N+1),159,(N*19)+ZZ*(N+1),ZZ:NEXT
   ZZ
90 FOR ZZ=1 TO 3:CIRCLE 10+N*35,60,
   20-ZZ*3,10+N*3,ZZ:NEXT ZZ
100 GOSUB 430:D=D+12
110 NEXT N
130 BLOCK 0,0,159,199,3
140 TEXT 0,100,"ANIMATION",0,8,19
150 D=64:FOR N=0 TO 4
170 GOSUB 440:D=D+12:FOR T=1 TO
   15:NEXT T
190 NEXT N
200 GOTO 150
220 FORZ=7680 TO 7738:READ X:POKE Z,X:
   NEXT Z:RETURN
230 DATA 169,0,141,14,220,169,53,133,1
240 DATA169,0,133,251,133,253,169,224,133,
   252,169,64,133,254,160,0
250 DATA 177,251,145,253,192,63,208,16,
   165,252,201,235,208,10
260 DATA 162,1,142,14,220,162,55,134,1,96,
   200
270 DATA 208,229,230,252,230,254,76,25,30
430 POKE 7700,D:POKE 7706,251:POKE
   7708,253:SYS 7680:RETURN
440 POKE 7700,D:POKE 7706,253:POKE
   7708,251:SYS 7680:
   RETURN
```

The program starts by setting the top of BASIC just enough below the start of Simons' to accept (note the CLR) a small machine-code routine which handles the paging of the graphics screens. This routine is loaded into memory by the subroutine at Line 220 and is later accessed for both storage and recall of the pages by SYS calls.

The first of the screen pointers is preset in Line 40 to the start of the RAM area available above Simons'. The program then continues by starting the drawing routine for the first of the five screens set up by the FOR. . .NEXT loop in Line 50. Lines 60 to 90 handle the actual design. When this is complete, a POKE routine in Line 440 is accessed, a SYS call is made to the machine-code routine, and the first picture block is confined to its appropriate place in memory.

The drawing loop then continues for a further four screens, each being allocated a fresh 3K block via the pointer adjustment D=D+12 in Line 100.

Line 130 then clears off the screen by overpainting in light blue, and Line 140 prints a message as a start to the animation sequence which follows immediately.



One method of appearing to use more pages in an animated sequence is shown above. By using only three separate images, two of which are repeated, a five-image, relatively smooth effect of walking is attained. With this technique, memory restrictions can be sidestepped

The memory pointer D is reset to the original 64 value, the start location of the first screen immediately above the end of Simons'. The first of the five screens is then recalled from memory using the routine in Line 440 which again accesses the machine code page graphics routine at location 7680.

The pointer is reset for the next screen (D=D+12 in Line 170), a small time delay loop is activated T=1 TO 15, and then the next screen is called up to overwrite the last. The program then cycles through the five frames, restarting with the original pointer when the loop is complete.

To make the program work on INPUT's Commodore hi-res program, repalce 224 in Line 240 by 32, and replace 235 in Line 250 by 43.

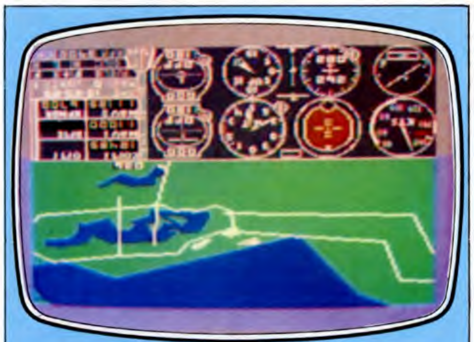
You can use the same machine-code routine for your own graphics displays, providing you establish the right-hand lower screen coordinate and set the low byte value in place of 63 and the high byte value in place of 253 in Line 250 when a Simons' cartridge is used. With the INPUT hi-res program in use, remember that the screen memory locations start at 8192 (lo 0, hi 32) and not 57344 as with Simons'. Otherwise the same two figures are changed.

If your animation design means that a smaller screen size can be employed, releasing more memory for paging, adjust the FOR. . .NEXT loops and the value of the D pointer increment to suit the number and memory size of each page.

Because of acute memory restrictions in many of the high resolution modes, it's best to restrict yourself to Mode 4 if you wish to explore the potential of paged graphics on the standard Acorn computers. The example program here uses nine paged screens of quarter screen depth.

In the program for the BBC that follows, use is made of a variation of the VDU23 instruction. As this is not on the Electron, a small machine-code routine has to be added and other amendments made before the program can be used on this machine. These follow the main listing. Even then, it's not possible to blank out the unwanted upper part of the screen which is used for screen image storage. The effect is nevertheless interesting.

```
10 MODE4:VDU23;8202;0;0;0;
20 B=PI/20
30 HIMEM=&2600:?&34E=26
40 VDU 23;6,8,0;0;0;
50 VDU 24,0;768;1279;1023;
60 FOR X=0 TO 8
70 PROCSCREEN
80 CLG
90 PROCDRAW
100 PROCWAVES
```



A practical application of paged graphics techniques in commercially available software is shown above: Flight Simulator II from subLogic, which runs on the Commodore 64. Here, smooth screen displays are achieved by treating the display as an animated film, the next frame being drawn off screen, and flashed on, completed

```
110 NEXT
120 FOR T=0 TO 10000:D=INKEY(5):
   X=T□MOD9:PROCSCREEN
130 IF T□MOD9=0 THEN SOUND1,-15,0,
   1
140 NEXT
150 END
160 DEF PROCSCREEN
170 MEM=&7600-X*&A00
180 ?&351=MEM□DIV 256
190 ?&350=MEM□MOD 256
200 MEM=MEM/8
210 *FX19
220 VDU 23;13,MEM□MOD&100,0;0;0;23;
   12,MEM□DIV&100,0;0;0;
230 ENDPROC
240 DEF PROCDRAW
250 MOVE1000,800:DRAW1000,1000
260 FOR T=800 TO 990 STEP16:MOVE1000,
   T:DRAW1016,T+16:NEXT
270 PRINTAB(33,3)"WALL"
280 MOVE500,875:DRAW450,875
290 DRAW450,925:DRAW500,925
300 DRAW500,875:DRAW575,825
310 MOVE500,925:DRAW575,975
320 PRINTTAB(3,2)"LOUDSPEAKER"
330 ENDPROC
340 DEFPROCWAVES
350 E=X*180
360 FOR G=1 TO 5
370 I=500+E*COS-B:IF I>1000 THEN
   I=2000-I
380 MOVE I,900+E*SIN-B
390 FORA=-B TO B STEP 0.1
400 I=500+E*COSA
410IF I>1000 THEN I=2000-I
420 DRAW I,900+E*SINA
430 NEXT
440 E=E+60
450 NEXT
460 ENDPROC
```

The initial choice of MODE 4, set in Line 10, is because this offers the best compromise between memory usage and resolution. A look here at the allocation of memory shows the reason for this. If one presumes a 'worst case' situation with a disk unit fitted, 3K is already accounted for. Add to this a realistic amount for the program and variables and about 6.5K is accounted for.

Restricting the display to a quarter depth screen in MODE 4, means that only 2560 bytes are required for each page. This gives you at most nine pages for animation purposes when the number of bytes is divided into the remaining free memory.

Nine screens is ample for the subject matter used as an example for this program. If you need a considerable extra amount of detail (hence more programming to draw it), you may find that the memory restricitions mean that one or more screens will have to be sacrificed. So, unavoidably, there's an element of suck-it-and-see involved in discovering how many are available.

THE PROGRAM

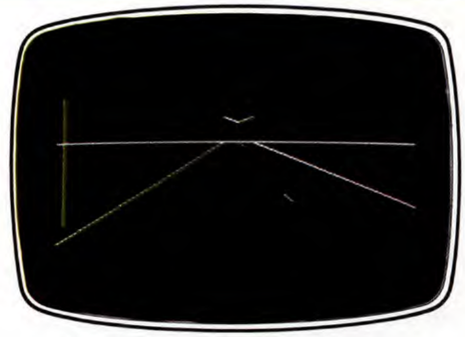
On with the program itself. Line 10 continues by introducing the first use of 23 in an instruction which simply 'loses' the cursor. The next line sets the variable used in one of the drawing routines later. The HIMEM instruction in Line 30 then sets the limit of BASIC and the start of screen page memory at &2600.

The VDU 23 instruction of Line 40 now sets the screen up to show the top eight lines only. Once you've got the program up and RUNning you could try removing this entire line. All four sectors of the screen become visible—the lower three displaying the stored graphics pages successively displayed in the top section.

The actual screen 'window' is set up by the VDU 24 instruction in the following line, the figure pairs afterwards representing the screen coordinate of the bottom left corner and top right corner respectively.

The PROC which follows works out the top left pixel position coordinates for each of the nine screens is established by Lines 180 and 190 and used in subsequent VDU 23 instructions. The *FX call in Line 210 simply delays the computer until the next 'frame' is ready to start.

The two VDU 23 instructions of Line 220 memorize the low and high byte values of each screen start address, using register 13 and 12 (respectively) of the 6845 chip. The remnant of each instruction is padded out with zeroes.



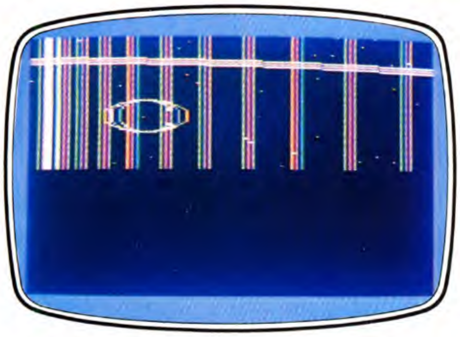
The Spectrum's endless road

Two separate PROCs follow as part of the graphics routine If you want to create your own paged graphics, you can use the main program without modification by inserting your own graphics routine in place of the example. But bear in mind the memory restrictions which limit the length of the drawing routines.

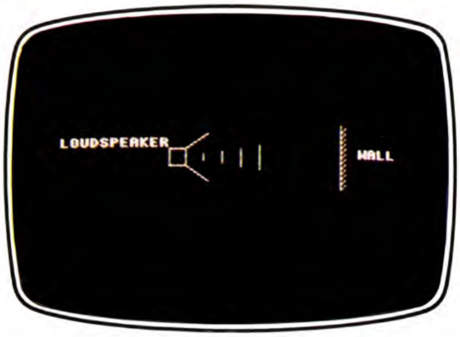
CHANGES FOR THE ELECTRON

Delete Lines 40 and 200 in the BBC program above and make the following changes for the Electron:

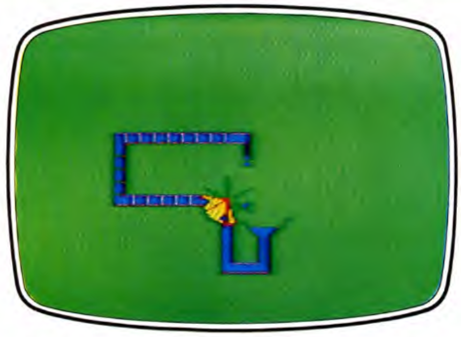
```
20 B=PI/20:PROCASS
50 VDU 24,0;768;1279;
   1023;:?&351=&76:
   ?&350=0
120 FOR T=0 TO 10000:D=INKEY(25):
   X= (T□MOD8)+1:
   PROCSCREEN
180 X%=MEM□DIV 256
190 Y%=&76
220 CALL SWITCH
470 DEF PROCASS
480 DIM SWITCH□50
490 FOR T=0 TO 2 STEP 2
500 P%=SWITCH
510 [OPT T
520 STX &71
530 STY &73
540 LDX #10
550 LDY #0
560 STY &70
570 STY &72
580 .L2□LDA (&70),Y
590 PHA
600 LDA (&72),Y
610 STA (&70),Y
620 PLA
630 STA (&72),Y
640 INY
```



Commodore screen graphics



Acoustics on the Acorn



Perpetual motion on the Dragon

```
650 BNE L2
660 INC &71
670 INC &73
680 DEX
690 BNE L2
700 RTS
710 ]:NEXT
720 ENDPROC
```

With these amendments, two chunks of memory attend to the necessary paging techniques. Line 5Ø sets the screen permanently to the &76ØØ–&8ØØØ block and then informs the computer that this is the case. The machine code is used to move the different screens in and out of the viewed screen which is the bottom eight lines on your TV. Pointers for the machine-code call are set in Lines 18Ø and 19Ø, where X% works out the high byte of the screen, a value of 1 to 8 established in Line 12Ø.

The Dragon and Tandy models have a clear-cut advantage over the other computers here because of the ready-made ability to handle paged graphics. This comes courtesy of the powerful PCOPY command used to shift graphics data from screen to memory and back again—a simple BASIC paged graphics command word!

In the example which follows, based on the choice of PMODE 3 graphics, five three-quarter size screens are used.

```
10 PCLEAR4:PMODE3:CLEAR40,9215
20 SCREEN1,0:FORK=0TO4:PCLS
30 CIRCLE(127,120),20,4,1,.17,.55:LINE(110,
   116)-(127,120),PSET:LINE-(132,136),
   PSET:PAINT(122,135),2,4
40 DRAW"BM137,136S8F6D10L9U15L4D19R1
   7U14E2NE6L8":PAINT(150,150),3,4:DRAW
   "C3BRR4C4"
50 DRAW"BM110,116S16L14U10R21D3RU5
   L24D14R15":PAINT(90,120),3,4
60 COLOR3:FORL=0TO5-K:LINE(148-L,
   146-L)-(156+L,146-L),PSET:NEXT
70 DRAW"BM141,"+STR$(86+K)+
   "C3S4F2G2H3E2D4":DRAW"BM141,"
   +STR$(INT(88+1.5*K*K))+"D2F2DL4
   UE2D4"
80 DRAW"BM"+STR$(INT(141
   +1.8*K))+","+STR$(127+5*K)+
   "H3E3F2G2DU4G2"
90 COLOR2:FORL=0TO5:LINE
   (110-L*10-K*2,117)-(110-L*10-
   K*2,123),PSET:NEXT
100 FORL=0TO7:LINE(54+L*10+K*2,
    75)-(54+L*10+K*2,69),PSET:NEXT
110 FORL=0TO3:LINE(48,115-L*10-K*2)
    -(52,116-L*10-K*2),PRESET:NEXT
120 IFK=0 THENDRAW"BM110,124C3H2UE2
    F2DG2U4"
130 COLOR4:FORL=0TO2:A=ATN(1)*
    (L*60-K*12)/45:LINE(127-18*SIN(A),
    120-18*COS(A))-(127+18*SIN(A),
    120+18*COS(A)),PSET:NEXT
140 A=ATN(1)*(8+K*12)/45:DRAW"BM"+
    STR$(INT(127-18*SIN(A)))+","+STR$
    (INT(120+18*COS(A)))+"C3E2UH2G
    2DF2U6C4"
150 FORL=2TO4:PCOPYL TO4+K*3+L:
    NEXTL,K
160 FORL=1TO5:FORK=2TO4:
    PCOPYK+L*3+1TOK:NEXT,L:GOTO160
```

The program starts by allocating four blocks (1.5K each) for screen data because PMODE 3 was chosen for this display and each screen in this mode requires 6K memory. Although four pages is the default value for PCLEAR, setting the value places the BASIC program exactly between the memory areas used for the screen and graphic data. In other circumstances the value can range from 1 to 8 depending on how much reserved memory is required.

So far 6K has been allocated. A further 1.5K is lost to BASIC and to the test screen, leaving about 25K of available RAM. Using whole screen pages requiring 6K memory apiece permits up to four screens of paged graphics (25K divided by 6)—but this leaves very little room for the program itself.

Restricting the display to just three-quarters of the screen depth means that each screen requires only 4½K memory. Five screens of page graphics can be accomodated, while leaving about 2K with the program itself. There isn't enough room for a disk operating system, however. After setting PMODE 3 display—four colours with a resolution of 128 × 192 pixels—the first line of the program CLEARs the 'meanest' amount of string storage space beyond memory location 9215, a figure that you may have to establish by trial and error for your own routines. The default, incidentally, is 2ØØ—which is rather wasteful under these circumstances.

The second line continues with the setting-up process by defining the screen resolution (hi-res) and colour set Ø, begins the graphics drawing loop, and ends with PCLS.

The graphics routines which follow occupy a significantly large part of the program. Line 3Ø builds and colours the body of the perpetual motion pump which forms the basis of the display. Line 4Ø constructs the funnel pipe and infills with colour. Line 5Ø does the same for the overhead pipe. Lines 6Ø, 7Ø and 8Ø look after the graphics for what is eventually an animated water drop sequence. Lines 9Ø, 1ØØ and 11Ø then construct the bottom, top and vertical water 'flow' stripes which also help suggest movement when the animated display is underway. Further pump detail—such as rotation of a paddle—are added by Lines 12Ø, 13Ø and 14Ø.

Line 15Ø then copies the bottom three-quarters of the screen into memory, into the protected area defined in Line 1Ø. The program loops to the start of the graphics routine creating an additional page of graphics on each pass, this too being confined to its appropriate memory location when Line 15Ø is reached again.

Once the graphic screens are in memory, the program proceeds with the page graphics routine handled by Line 16Ø. This simply copies what's been copied into memory back to the screen, in a five-screen-loop which creates the animation sequence.

MUSIC IN ENVELOPES-COMMODORE/ACORN

When you master the envelope statement on your Commodore 64 or Acorn micro, you can mimic a vast range of sounds and music—useful for your other programs

Adding sound can make all the difference to an ordinary, run-of-the-mill program and make it an interesting one that is exciting, informative and fun to use. Most microcomputers allow you to program pure notes or noises for applications such as games, simple tunes and special effects. For more sophisticated work, however, it is much better if you can modify the tones generated by the micro to enable you to mimic sounds—from an emergency siren to a chirping bird or a particular musical instrument. This facility is provided directly from BASIC by the ENVELOPE statement, which is available on Acorn micros and indirectly (by POKEing memory locations) on the Commodore 64.

WHAT IS A SOUND ENVELOPE?

An electronic sound is produced by a circuit called an oscillator. This generates a wave of a particular frequency (pitch) and amplitude (volume). When this is passed to a loudspeaker a note of that frequency and amplitude sounds. To change the note, vary its two parameters—higher frequencies give higher pitch, and greater amplitude gives more volume. Some computers do not give you this much control—the Spectrum, for example, only lets you vary the frequency (and the duration), not the volume of the note.

Changing these parameters generates a wide range of musical notes. If you combine all the notes at once, you produce a noise of indeterminate pitch—called white noise. This is the principle of the SOUND command on the BBC micro, for example, but it produces only a limited range of sound effects. This is because the note it produces is purely mechanical without the characteristics of any particular instrument—the quality that makes a saxaphone playing middle C sound different from a piano playing the same note. To mimic the sound of a piano or an organ, for example, the notes generated by the oscillators in the micro must be modified—the waveform of the notes must be shaped. This is what a

- WHAT IS A SOUND ENVELOPE?
- WAVE MODULATION
- THE AMPLITUDE ENVELOPE
- THE PITCH ENVELOPE
- DESIGNING A SOUND
- HOW IT WORKS
- SHAPING THE WAVE
- INTERDEPENDENCE OF PHASES
- MAKING SOUNDS
- BBC AMPLITUDE ENVELOPES

synthesizer does to enable it to produce a wide range of sounds.

The shaping of a sound wave is a form of modulation—the same principle that makes it possible for speech to be transmitted by radio waves. Radio starts with a wave of particular frequency and amplitude (a carrier wave) on which is superimposed the speech waveform, which envelopes the sine-wave pattern of the carrier wave. If the envelope shapes the peaks of the carrier wave, the amplitude is no longer constant, but varies according to the speech wave—this is Amplitude Modulation or AM. If the envelope shapes the carrier wave along its length (in time), the frequency of the carrier increases or decreases according to the speech wave to give Frequency Modulation or FM.

In a similar way, micros can have amplitude envelopes (found in the Commodore 64), or frequency envelopes—usually called pitch envelopes—found on the Electron micro. The BBC micro is unusual—it has both types of envelope. The effects of either type of modulation are to impose a new and subtly different quality onto the pure note.

However sophisticated the micro, its sound producing quality is unlikely to be as good as that of a synthesizer—which is dedicated to producing sounds. So it is difficult for a computer to generate a convincing synthesis of acoustic instruments. Imaginative use of envelopes, however, can let you produce a wide range of sounds, some are reasonable mimics of real instruments and others totally unlike any existing instruments, but nonetheless interesting and useful.

THE AMPLITUDE ENVELOPE

The easiest way to understand an amplitude envelope is with a graph of loudness or amplitude against time. When a musical instrument sounds a note, energy is applied to the string, reed or whatever, making it vibrate and setting the surrounding air in motion.

The sound of an organ, for example, rises quickly to a peak of loudness, which is maintained at a constant level while the note is sounded, then it dies away. On a piano or guitar, however, the initial rise is similar to the organ's, but the loudness instantly begins to die away. If the key is then released (or the string touched), the sound is silenced quickly. To simulate these sequences of the way in which a note builds up, is sustained and then falls away, you can program an envelope as part of the SOUND command on the Acorns, or the MUSIC and PLAY on the Commodore.

THE PITCH ENVELOPE

The pitch envelope is a little more complicated to understand than the amplitude envelope. Once again, it can be depicted as a graph against time, in which the frequency speeds up or slows down, raising or lowering the note. Shaping the envelope, however, is not the same as simply changing a note to a different pitch, since the variation occurs in a controlled and regular way over a period.

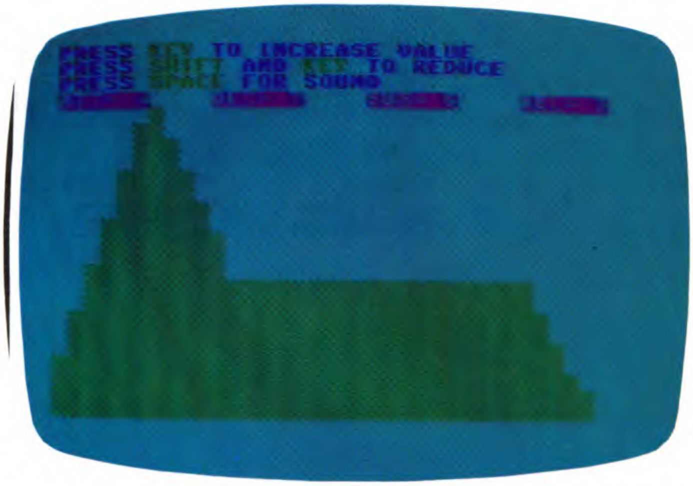
An example of how this might appear in music is when vibrato is applied to a note, producing a throbbing effect. Or in sound effects, it could be the note produced by a siren, warbling up and down on a regular basis. Another common example is the 'Doppler effect', in which the sound of a fast moving vehicle appears to rise in pitch as it comes towards the listener, falling again as it moves away.

The pitch envelope is only available on the Acorn machines. On the BBC, it can be used in conjunction with the amplitude envelope to produce some of the most sophisticated synthesized sounds that are available on a microcomputer.

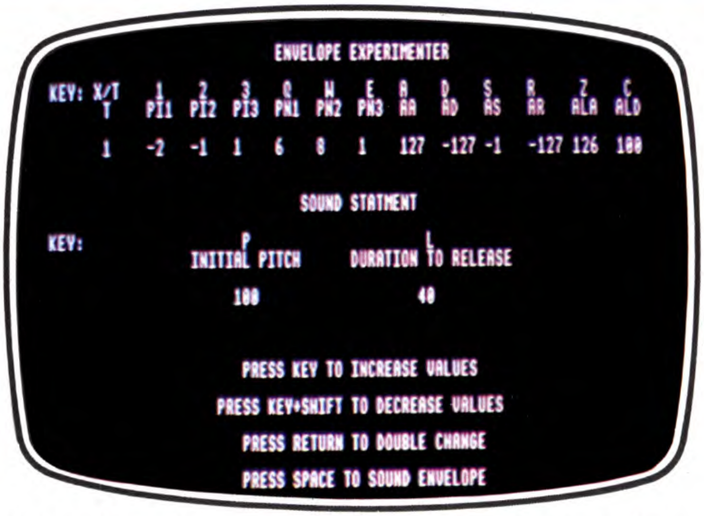
DESIGNING A SOUND

Although you can soon learn to analyse certain sounds, it is not always easy to design an envelope to give the effect you require, so the task, especially for creating sound effects, is often a matter of trial and error. Here is a program that lets you change the envelope parameters, then listen to the sound:

```
10 POKE650,128:POKE 53280,3:POKE
   53281,3
20 A=4:D=7:S=6:R=9:FF=17
30 FOR I=1 TO 20:S$=S$+"▩○▌"
   :NEXT
40 RI$="▟▟▟▟▟▟▟▟▟▟
   ▟▟▟▟▟▟▟▟▟▟▟▟▟
   ▟▟▟▟▟▟▟▟▟▟▟▟▟
   ▟▟▟▟▟"
50 BOT$="▤▙▙▙▙▙▙▙▙▙
   ▙▙▙▙▙▙▙▙▙▙▙▙
   ▙"
60 GOSUB 1000
70 POKE 54296,9
80 POKE 54273,40
100 GET X$
110 IF X$="□" THEN GOSUB 500
120 IF X$<>"" THEN GOSUB 1000
130 GOTO 100
500 POKE 54276,FF
510 FOR I=1 TO 230:NEXT
520 GET X$:IF X$="□" THEN 510
530 POKE 54276,FF-1
```



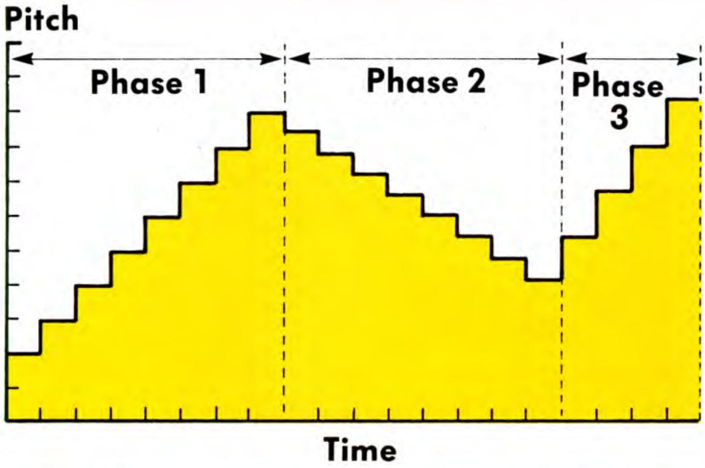
An amplitude envelope produced on the Commodore 64

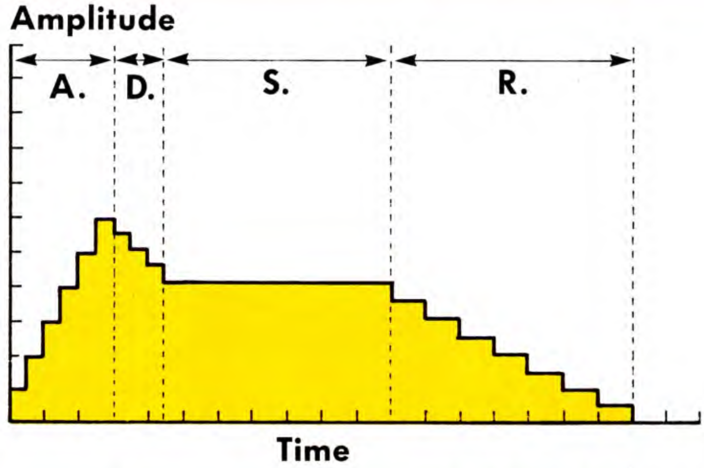


At least 18 variables must be set to specify a BBC envelope

```
540 RETURN
1000 IF X$="A" THEN A=A+1
1010 IF X$="D" THEN D=D+1
1020 IF X$="S" THEN S=S+1
1030 IF X$="R" THEN R=R+1
1040 IF X$="♠" THEN A=A-1
1050 IF X$="[graphic]" THEN D=D-1
1060 IF X$="[graphic]" THEN S=S-1
1070 IF X$="[graphic]" THEN R=R-1
1076 IF X$="[graphic]" THEN FF=17
1077 IF X$="[graphic]" THEN FF=33
1078 IF X$="[graphic]" THEN FF=65
1079 IF X$="[graphic]" THEN FF=129
1080 A=A AND 15:D=D AND 15:S=S AND
   15:R=R AND 15
1090 POKE 54277,A*16+D
1100 POKE 54278,S*16+R
1120 PRINT "[graphic]ATT="A,
   "DEC="D,"SUS="S,"REL="R
1130 PRINT"[graphic]PRESS [graphic]KEY[graphic] TO
   INCREASE VALUE"
1140 PRINT"PRESS [graphic]SHIFT[graphic] AND
   [graphic]KEY[graphic] TO REDUCE"
1150 PRINT"PRESS [graphic]SPACE[graphic] FOR
   SOUND[graphic]"
1200 XX=0:HT=0
1210 HT=HT+(16-A)/4:IF HT>20 THEN
   HT=20
1220 GOSUB 2000
1230 IF (HT<20) AND (XX<40) THEN 1210
1240 IF XX=40 THEN RETURN
1250 HT=HT-(16-D)/4:IF HT<20*S/15
   THEN HT=20*S/15
1255 IF HT<1 THEN HT=0
1260 GOSUB 2000
1270 IF (HT>20*S/15) AND (XX<40) THEN
   1250
1280 IF XX=40 THEN RETURN
1290 FOR X=XX TO 30:REM SUSTAIN
   PHASE
1300 GOSUB 2000
1310 NEXT
1320 HT=HT-(16-R)/4:IF HT<0 THEN
   HT=0
1330 GOSUB 2000
1340 IF (HT>0)AND (XX<40) THEN 1320
1350 RETURN
2000 PRINT BOT$
2005 IF HT=0 THEN RETURN
2010 PRINT LEFT$(RI$,XX);
2020 PRINT LEFT$(S$,HT*3+1)
2030 XX=XX+1
2040 RETURN
```

The Commodore 64 allows you only to shape the amplitude envelope of the basic wave—you have a choice of four types of wave to start with. Then, to synthesize a particular type of sound, you can program four phases—Attack (A), Decay (D), Sustain (S) and Release (R). This type of envelope is usually called the ADSR system. The Attack phase is the rate at which the loudness increases when the sound is first initiated. The Decay is the initial dieing away, then comes the Sustain—the steady level, as in an organ. The fourth phase—the Release—is the rate at which the





loudness falls to zero.

Run the program to see initial ADSR values printed at the top of the screen, and a graph of the envelope produced by these values. Press the space bar for a note.

You can change the ADSR in this program by pressing the appropriate key: hold down A, D, S or R to increase the values; SHIFT with the same key to decrease the values. After each keypress, the printed values and the graph will be updated on the screen.

The type of sound produced by any set of ADSR values depends on the basic waveform that the envelope is shaping. The program lets you change between the four available, using the function keys. Key f1 gives triangle, f3 gives sawtooth, f5 gives pulsed and f7 gives noise. For certain settings, you may not detect a sound when you press f5, because the sound decays before reaching an audible level.

When you come to program your own sounds, it is useful to understand how this program works, but in practice you generally need only a few lines of program to make fairly complicated sounds. Line 1Ø enables auto-repeat of the keys, to detect that the space bar is being held down. Line 2Ø sets the initial ADSR values. Line 3Ø sets up a character string to draw the graph (a histogram) of the envelope. Line 4Ø sets up a string to move the cursor to the bottom of the screen, and Line 5Ø sets up a string to move the cursor a controlled distance along the bottom of the screen, placing the cursor on the column for the current histogram block.

Line 6Ø calls a subroutine to display the envelope for the intital ADSR values. Lines 7Ø and 8Ø set the master volume and frequency of the oscillator for voice 1. Volume settings can range from Ø (off) to 15 (loudest), so you can experiment with this by changing the second figure at Line 7Ø. Lines 1ØØ to 13Ø form the main loop: the program waits for a key, updates the ADSR values and displays the revised histogram.

The subroutine at Line 5ØØ ensures that envelope one is switched on so long as the space bar is pressed. The subroutine at Line 1ØØØ updates the ADSR values (Lines 1ØØØ to 1Ø8Ø), and POKEs them into the appropriate registers (Lines 1Ø9Ø to 11ØØ). Lines 12ØØ to 135Ø calculate the histogram, working through the four phases (A, D, S, R). The variables XX and HT are the X coordinate and height of the current column.

The subroutine at Line 2ØØ displays the column of HT at position XX, using the strings mentioned earlier. The cursor is placed at the bottom of the screen, at column XX, by Lines 2ØØØ and 2Ø1Ø. The column is then drawn by Line 2Ø2Ø.

SHAPING THE WAVE

The ADSR phases for voice 1 are each controlled by nybbles in location 54277 and 54278: the attack value is controlled by the upper nybble of location 54277, the initial decay by its lower nybble, the sustain level by the upper nybble of 54278 and the final release by its lower nybble.

For an oscillator to become audible, a waveform must be selected and its envelope shaper triggered: this is done by setting bits in the register that controls the voice (at location 54276, Line 5ØØ, for voice 1). Bit Ø of this register is the 'gate' or 'trigger' for the voice's envelope shaper. Its action is rather like that of a key on an organ: as long as a key is held down the note sounds. When the key is released the note enters its final release phase.

The control registers also select and switch on the waveform to be used: setting bit 7 to 1 selects randomnoise, bit 6 selects pulse wave, bit 5 selects sawtooth wave, bit 4 selects triangular wave. The bit controlling the envelope has to be combined with the bit selecting waveform. So POKE 54276,33 selects a sawtooth wave and switches the gate on: POKE 54276,32 switches the gate off (by setting bit Ø to Ø) but leaves the waveform selected and thus initiates the envelopes's final release phase. Note that POKE 54276,Ø would switch the gate off but would also deselect the waveform, silencing the voice.

INTERDEPENDENCE OF PHASES

To make the best use of the ADSR envelope, it is important to understand how the four phases interact. Consider what happens if the Sustain value is 15, its maximum value. The maximum level (reached at the end of the attack phase) will be the same as the sustain level, so there is no decay from one to the other, and the resulting envelope will sound the same irrespective of the decay value.

As a second example, suppose the sustain value is Ø, initial decay is quite long and final release is very long. At the end of the attack phase, the note starts to decay towards nothing at the initial decay rate. Switching the gate off sooner rather than later, (but before zero level has been reached), will, rather paradoxically, extend the note, because the slower-falling release phase is entered.

```
10 MODE0
20 VDU28,0,31,79,3,23;8202;0;0;0;
30 N=16
40 DIM I(N),A(N),MIN(N),MAX(N)
50 FOR T=1 TO N
60 READ I(T),A(T),MIN(T),MAX(T)
70 NEXT
80 PRINT TAB(27,0)"ENVELOPE
   EXPERIMENTOR"
90 PRINT TAB(4,2)"KEY:□X/T□□□□1□
   □□□2□□□□3□□□□Q□□□
   □W□□□□E□□□A□□□□D□
   □□□S□□□□R□□□□Z□□□
   □C"
100 PRINTTAB(10,3)"T□□□□PI1□□
   PI2□□PI3□□PN1□□PN2□□PN3
   □□AA□□□AD□□□AS□□□AR
   □□□ALA□□ALD"
105 PRINTTAB(26,17)"PRESS KEY TO
   INCREASE VALUES"TAB(23,19)"PRESS
   KEY+SHIFT TO DECREASE VALUES"TAB
   (26,21)"PRESS RETURN TO DOUBLE
   CHANGE" TAB(26,23)"PRESS SPACE TO
   SOUND ENVELOPE"
110 PRINT TAB(33,8)"SOUND STATEMENT"
120 PRINT TAB(4,10)"KEY:"TAB(26,10)"P"TAB
   (48,10)"L"TAB(20,11)"INITIAL□PITCH
   □□□□□□□
   DURATION□TO□RELEASE"
130 IF INKEY(-1) THEN D=-1 ELSE D=1
140 IF INKEY(-74) THEN D=D*2
150 FOR X=1 TO N:PROCJ:NEXT
160 ENVELOPE1,A(1)+A(2)*128,A(3),A(4),
   A(5),A(6),A(7),A(8),A(9),A(10),A(11),
   A(12),A(13),A(14)
170 PROCSHOW
180 IF NOT INKEY(-99) THEN 130
190 SOUND17,1,A(15),A(16)
200 REPEAT UNTIL NOT INKEY(-99):GOTO
   130
210 DEF PROCSHOW
220 PRINTTAB(10,5);A(1)+A(2)*128"□□"
230 FOR T=3 TO 14:PRINTTAB(T*5,5);A(T)
   "□□□":NEXT
240 PRINTTAB(25,13);A(15)"□□"TAB
   (47,13);A(16)"□□"
250 ENDPROC
260 DEF PROCJ
270 IF INKEY(-I(X)) THEN A(X)=A(X)+D
280 IF A(X)>MAX(X) THEN A(X)=MIN(X)
290 IF A(X)<MIN(X) THEN A(X)=MAX(X)
300 ENDPROC
310 DATA 36,1,0,127
320 DATA 67,0,0,1
330 DATA 49,-2,-128,127
340 DATA 50,-1,-128,127
350 DATA 18,1,-128,127
360 DATA 17,6,0,255
370 DATA 34,8,0,255
380 DATA 35,1,0,255
390 DATA 66,127,-127,127
400 DATA 51,-127,-127,0
410 DATA 82,-1,-127,0
420 DATA 52,-127,-127,0
430 DATA 98,126,0,126
440 DATA 83,100,0,126
450 DATA 56,100,0,255
460 DATA 87,40,1,254
```

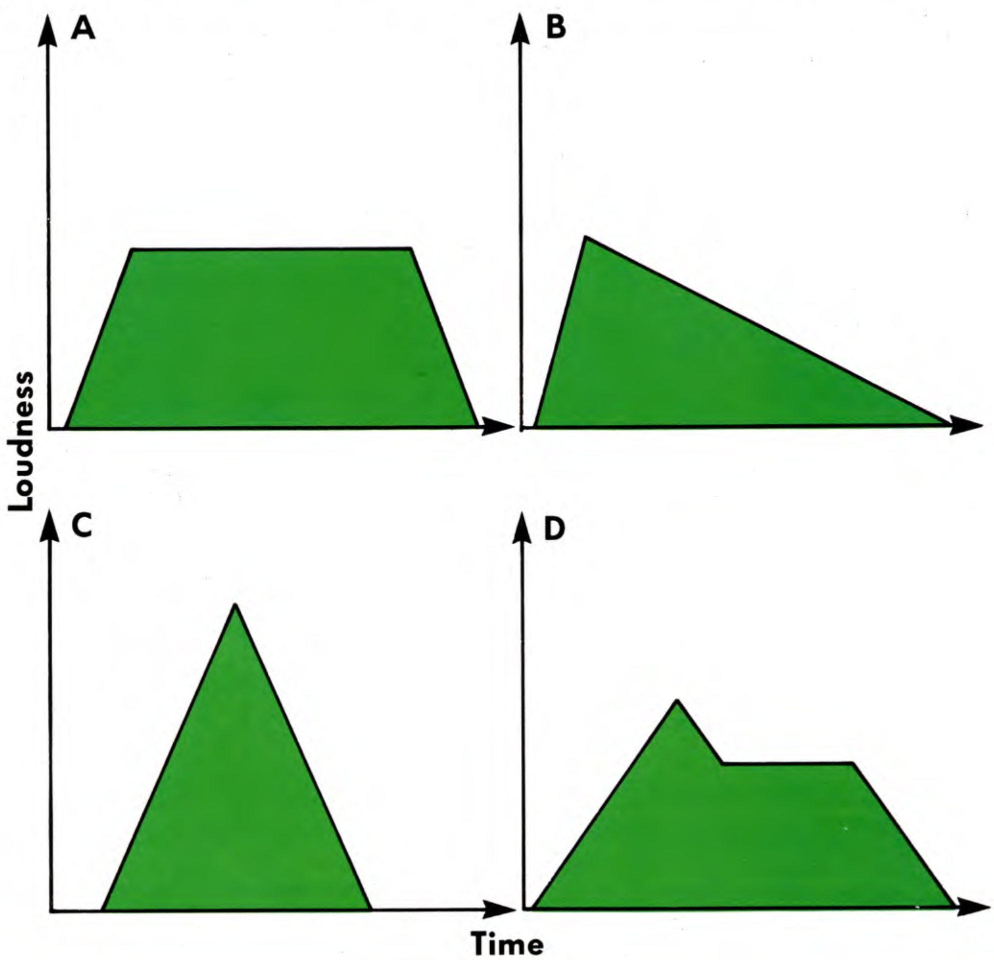
This program allows you to shape either a pitch envelope or an amplitude envelope. Users of the Acorn Electron can enter the program without modifications, but this micro does not allow you to vary the last six parameters of the sound statement, so you can specify only pitch envelopes. For that reason the amplitude envelopes section of this article applies only to the BBC micro.

Enter and RUN the program, when the display will show the keys you can press to change parameters in this program. Beneath these appear the variable names used in the User Guide (page 182) to define the ENVELOPE command, and beneath these are the initial values. The program uses four arrays (dimensioned at Line 40) to READ 16 lines of DATA into a FOR...NEXT loop (Lines 50 to 70). Each line of DATA consists of a value to identify the key being pressed, the initial value for each envelope parameter and the maximum and minimum values this parameter can take.

Lines 80 to 120 PRINT the screen display, and Lines 130 and 140 set intervals by which the parameters can change. If you press any of the keys listed on the screen, Line 130 lets the parameter for that key increase by one, but if you press SHIFT as well, then the parameter decreases by one. Line 140 detects RETURN, which you can press to increase or decrease the parameters—to speed up the program.

Line 150 updates the parameters, using a routine (Lines 260 to 300) to ensure that values change continuously from maximum

Pitch and amplitude envelopes can be designed separately (far left) on the BBC micro, but they are specified as a single statement. The four phases (D) of an amplitude envelope need not all occur in every type of sound. Some sounds (A) have no decay; others (B and C) have neither decay nor release



to minimum, at the rate set by D. Using these values, Line 16Ø calls the ENVELOPE command, which has 14 parameters. Line 17Ø calls a routine to print the new values on the screen. The first value in the ENVELOPE command is always the envelope number—in this case 1. Normally, you can define envelopes 1 to 4, but if you do not use the BASIC statement BPUT and BGET you can define 1 to 16.

The first updated value in the envelope command to be printed (Line 22Ø) is the time interval, which can vary between 1 and 127. Normally, the pitch envelope auto-repeats until the sound dies away. If you don't want the auto-repeat, add 128 to the value of the time interval to get a single execution.

Line 23Ø prints the other updated values in the envelope command, and Line 24Ø prints two values in the SOUND statement (called at Line 19Ø). Line 18Ø detects whether the space bar is pressed. If it is, the SOUND statement calls the envelope to make the sound.

Besides the envelope number, the SOUND statement sets values for initial pitch and duration of the sound, and the first parameter contains an equally important element. Normally, you can have a sound statement with channels Ø, 1, 2, or 3, but notice that Line 19Ø has 17 as the channel number. In fact, the channel number comprises four parameters (see page 187 of the User Guide). The first two of these control when notes are sounded on each of the four channels; the second can be Ø or 1, and controls whether a note is placed in the queue or whether other notes are flushed so that a particular note sounds immediately. In this program, we need notes to sound instantly, so the sound statement is flushed (set to 1). If the first two paramemters are not set, they can be ignored (they are zeros), so a value of &11 specifies immediate sounding on channel 1. In decimal, this is 17, which explains the unusual value in Line 19Ø.

MAKING SOUNDS

Change the values displayed on the screen, by pressing the keys indicated, and press the space bar to make the sound you have defined. Notice that you can press all the keys at once, including RETURN to double the rate of change. It is hard at first to predict what type of sound each set of values produce, but you should begin to do so once you have understood the parameters.

The program is initialized with data that specify a combined envelope—after the envelope number and time interval (the first two parameters), the next six values are those that would be varied to specify a pitch envelope. The last six parameters can be changed to specify a sound whose amplitude varies in time, but whose pitch remains constant. This pitch is set by the third value in the sound statement.

BBC AMPLITUDE ENVELOPES

The type of sound specified by an amplitude envelope can be plotted as a graph with four phases—Attack, Decay, Sustain and Release. These are abbreviated to AA, AD, AS and AR. During the Attack, the amplitude changes in steps, the number of which is controlled by the fifth parameter in this section of the envelope statement—the target level at the end of the Attack phase (ALA). The Decay is the change of amplitude per step, and again there is a target level (ALD) at the end of this phase. After these two phases, there comes the Sustain, which is the change of amplitude per step. This time, there is no separate parameter to limit the duration of the sustain; instead it lasts from the end of the decay to the start of the Release. The Release is also a change of amplitude per step—the number depending on how long the amplitude takes to decrease to zero.

The pitch envelope (also available on the Electron) also changes in steps, but it has only three phases—unlike the amplitude envelope's four. The six parameters that specify this type of envelope are PI1, PI2, PI3 (the change of pitch per step in each of the three sections), PN1, PN2 and PN3 (the number of steps in each section). A typical sound generated by a pitch envelope might be an emergency siren, which increases from a value set by the sound statement, then repeatedly decreases and then increases.

The sophistication of the BBC micro's sound handling qualities becomes apparent when you combine both types of envelope—pitch and amplitude. This lets you shape sounds that are complicated to analyse, and is one of the reasons the task of specifying envelopes is best aided with a program such as the one listed here that lets you see, as well as hear, the results.

Microtip

Combined envelopes—BBC

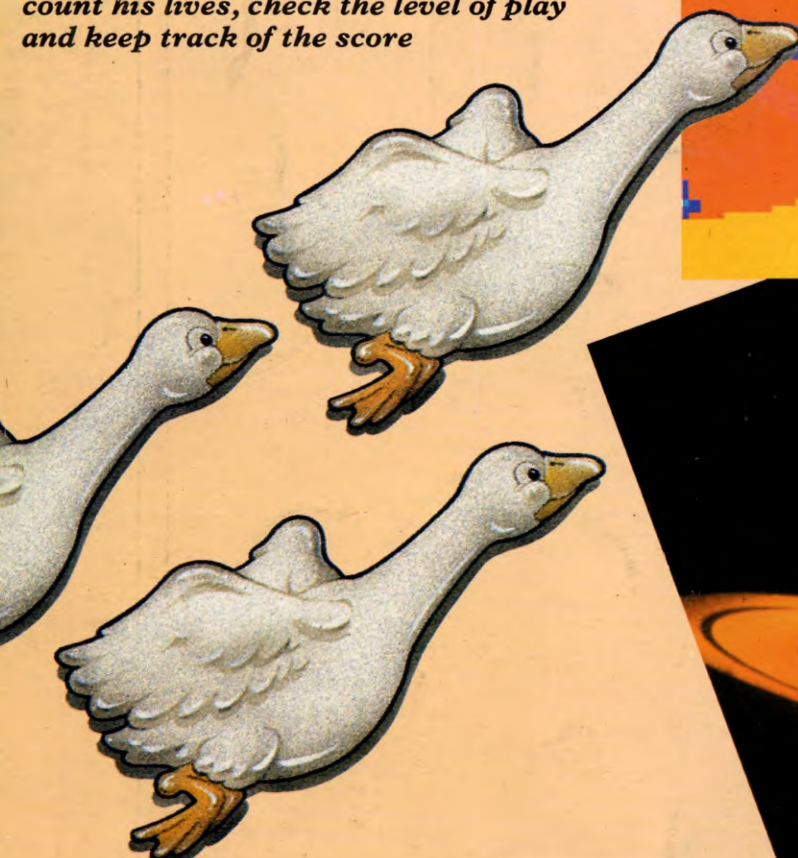
A single envelope statement on the BBC specifies a combined pitch and amplitude envelope, but it is sometimes possible to have one without the other. The envelope number and time period, followed by zeroes for each of the next six parameters, specify an amplitude envelope—provided suitable values are given to the last six parameters of the statement. If however, you design a pitch envelope, you must set a minimum value for ALD, otherwise the sound will not be audible. Notice also that if you set AR to zero, then any envelope you sound will be continuous. You can stop the sound by simply setting AR to any value.

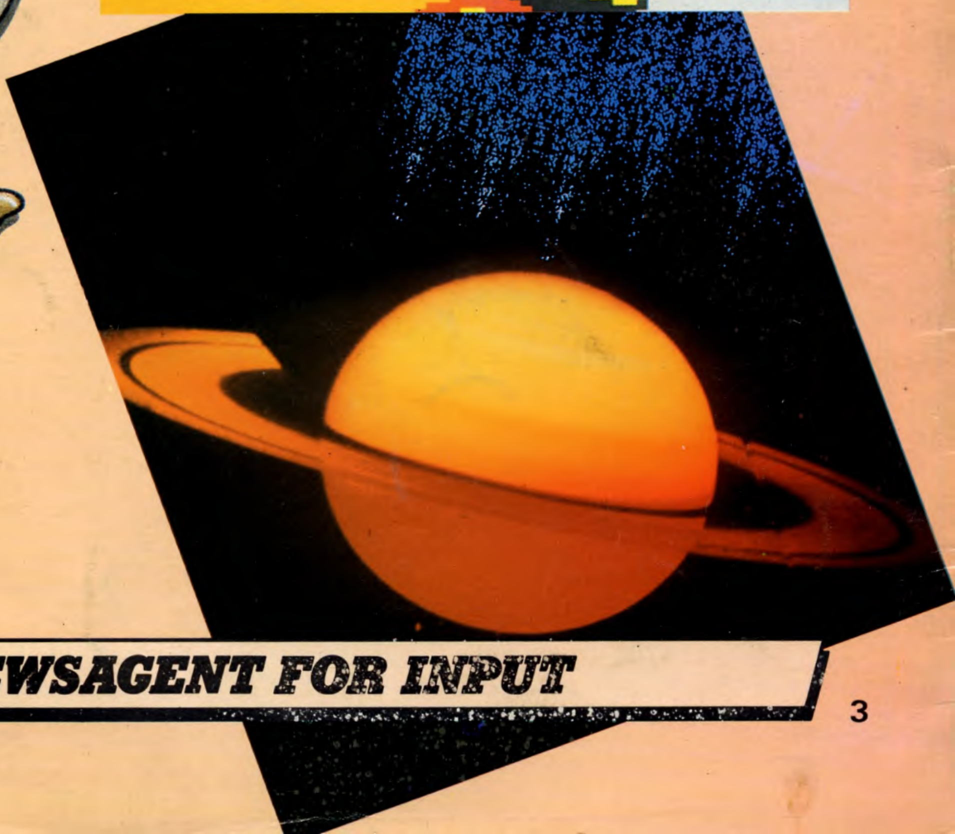
CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT.*

COMING IN ISSUE 37 . . .

- *Add the final programming to FOX AND GEESE and see how hard it is to get your aggressive gaggle to corner the intrepid fox*
- *If you think modelling means parading down a catwalk, try* INPUT'*S COMPUTER PREDICTION METHODS*
- *Use mathematical equations derived from the behaviour of natural phenomena to create unusual high-resolution PATTERNS*
- *Bespoke financial planning can be yours when you look into the possibility of TAILORING SPREADSHEETS*
- *What's Willie up to in this section of CLIFFHANGER? Find out when you count his lives, check the level of play and keep track of the score*





ASK YOUR NEWSAGENT FOR INPUT